ENHANCEMENTS TO CHEBYSHEV-PICARD ITERATION EFFICIENCY FOR GENERALLY PERTURBED ORBITS AND CONSTRAINED DYNAMICAL SYSTEMS

A Dissertation

by

BRENT DAVID MACOMBER

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	John L. Junkins
Committee Members,	Aniruddha Datta
	John E. Hurtado
	James D. Turner
Head of Department,	Rodney Bowersox

August 2015

Major Subject: Aerospace Engineering

ABSTRACT

Modified Chebyshev Picard Iteration (MCPI) is an iterative numerical method for solution of Ordinary Differential Equations (ODEs). This dissertation presents a body of work that serves to enhance the overall performance and the algorithmic automation of MCPI, applied to the problem of perturbed orbit propagation. Additionally, an MCPI framework is derived that greatly improves MCPI performance for ODE systems that intrinsically have associated conserved quantities. Leveraging these developments, software libraries are presented that are designed to make MCPI more accessible and automated, both for the problem of orbit propagation, and for general ODE systems. The work outlined in this document is the result of an effort to promote MCPI from an algorithm of academic interest to a broadly applicable toolset for general use by researchers worldwide in all disciplines.

MCPI is able to numerically propagate perturbed orbits to arbitrarily high solution accuracy, bounded by the limits of numerical precision. The improvements to MCPI for orbit propagation are focused on decreasing the computational cost of high-accuracy propagation in a two-fold manner; by reducing the number of required iterations necessary to achieve convergence, and decreasing the computational cost per iteration. Typically, the spherical harmonic gravity function evaluations are the most computationally expensive part of perturbed orbit propagation, so the strategies for reducing the cost per iteration focus on techniques for reducing the cost of gravity series evaluations. Additionally, automated tuning parameter selection logic is introduced to enable MCPI to propagate large batches of perturbed orbits, without the necessity of a user in the loop.

By making use of an associated conserved quantity within applicable ODE sys-

tems, MCPI is shown to be able to achieve much higher performance. A first order and a second order constrained MCPI formulation are developed that are able to vastly reduce the required number of iterations for convergence, increase the achievable segment length, and increase the overall solution accuracy for a given convergence threshold.

Software libraries are presented with the goal of encouraging widespread use of the MCPI method. Serial libraries are available for general ODE systems, akin to the Matlab ODE** methods. More specialized libraries, making use of the computational improvements and automated tuning, are available for perturbed orbit propagation. A parallel framework based upon the orbit propagation libraries is presented that is designed for space catalog maintenance, uncertainty propagation, or conjunction analysis.

Dedicated to my parents, for their unconditional support.

And for teaching me the way of *aretê*.

ACKNOWLEDGEMENTS

Thanks are due to many people, without whom this dissertation would never have been possible.

First and foremost, Dr. John L. Junkins. Thank you for all of your invaluable guidance and support over the course of my graduate school journey. Your seemingly endless wealth of knowledge is awe inspiring. I can honestly say, regardless if the conversation was a highly technical debate on a tricky research issue, or a discussion on the proper technique for downhill skiing, I came away from every conversation with you having learned something useful. I hope that I am able to emulate, at least in some small part, your research and problem solving style and your drive for knowledge.

My committee members, Dr. Aniruddha Datta, Dr. John E. Hurtado, and Dr. James D. Turner. Thank you for your mentorship, and for giving me the motivation to continue developing and improving this work. Your passion for research excellence is inspiring!

To my colleagues and collaborators, both from the LASR Lab and the MCPI research team. Thank you for the collaboration and the constant drive to learn and improve. Manoranjan Majji, Xiaoli Bai, Jeremy Davis, James Doebbler, Ahmad Bani-Younes, Donghoon Kim, Kurt Cavalieri, Clark Moody, Robyn Woollands, Austin Probe, Dylan Conway, Julie Read, and Abhay Masher. You guys are the best, I am privileged to have been able to work with you all.

To the administrative heroes in the Aerospace Engineering department and TIAS. I could never have navigated the hurdles of graduate school without you. Lisa Jordan, Karen Knabe, Rose Sauser, Yolanda Veals, and Rebecca Marianno, thank you very

much for everything!

And last, but certainly not least, my family and friends. Thank you for your unbounded support and understanding. For being there to celebrate during the good times, and being there to commiserate during the bad times. This endeavor was only possible because of the knowledge that you guys have my back.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

1.1 Motivation

Currently the U.S. government maintains a space catalog containing orbit estimates for about 20,000 space objects, the vast majority of which are uncontrolled debris objects. The current radar and optical sensing network is able to detect and track objects roughly the size of a basketball and larger. The planned next-generation Lockheed Martin "Space Fence" radar network[1] will be able to detect and track objects roughly the size of a golf ball, which will increase the size of the space object catalog to 100,000 objects or more[2] when the new sensor network comes online in 2017 [1].

An order of magnitude increase in catalog size translates to many orders of magnitude increase in computational burden for catalog management. Objects detected by the space fence must be tasked for follow-up observations by other sensors, and must be identified as an object already existing in the catalog, or else classified as a new object as yet unknown to the catalog. Follow-up observations must be incorporated into the current catalog orbit estimates using an iterative least squares method, a step called catalog maintenance. Furthermore, the current best orbit state estimates of all catalog objects must be used to determine if any two sets of objects pose a potential risk of collision in the future, a step called conjunction analysis.

As the number of objects in space increases, the probability of collision between objects increases. Collisions between objects, e.g. the Iridium/Cosmos collision of 2007, cause a sharp spike in the number of debris objects, further increasing the

[1]More information may be found on the Lockheed webpage regarding the space fence: http://www.lockheedmartin.com/us/products/space-fence.html

[2]According to the infographic on the Lockheed space fence webpage, the expected size of the catalog could increase to 200,000 objects or more when the new sensor network comes online.

probability of more collisions, leading to a situation that can ultimately devolve into debris clouds surrounding the Earth caused by cascading collisions (an undesirable situation called the Kessler Syndrome) [2]. The long-term remedy for preventing this from occurring is to reduce the number of uncontrolled debris objects in space. In the short-term, accurate catalog maintenance and conjunction analysis is the best prevention method for unintended collisions involving controlled satellites, such as the Iridium/Cosmos incident. Computationally, the most resource intensive process is the calculation of high precision orbit predictions, a necessary subroutine in all aspects of catalog maintenance and conjunction analysis (from sensor tasking and tracking, catalog identification, orbit maintenance, and ultimately conjunction analysis). These predictions come from numerically integrating the equations of perturbed orbital motion, a process called orbit propagation. As the number of objects in space increases, and thus the number of trackable objects in the space catalog increases, the computational burden of numerical propagation will outpace the computational resources available. New, more efficient, numerical propagation methods can alleviate the burden on the (already strained) computational resources available for catalog maintenance and conjunction analysis. New parallel processor architectures can be leveraged by a new class of parallelizable numerical propagator methods, allowing the inclusion of computationally more expensive (but more highly correlated to reality) perturbation force models, or higher-precision uncertainty characterization [1].

1.2 Background

Historically significant numerical propagators for solving the equations of perturbed orbital motion broadly fall into two classes: single-step methods (Runge-Kutta methods), and multi-step methods (predictor-corrector methods). The basis of a single-step method is, from a point on the state-space trajectory at a given time

$\boldsymbol{x}(t_k)$, the value of the state at a later time $\boldsymbol{x}(t_k+h)$ may be found by linearly combining weighted evaluations of the ODE function at intermediate times $t_k \leq t \leq t_k + h$. The method is called single-step because only information that is extrapolated from the initial state value $\boldsymbol{x}(t_k)$ and forward is used to find the later state value $\boldsymbol{x}(t_k+h)$. Contrastingly, multi-step integration methods estimate the later state value using the current value, and several previous state values. Generally, the later state value is extrapolated from the previous values in a forward estimation step (called the prediction step), and then refined in a backwards estimation step (called the corrector step) [3]. The predictor and corrector cycles are iterated until an error criterion is satisfied.

Explicit Runge-Kutta methods are a family of single-step integrators that utilize simple weighted linear combinations of evaluations of the ODE function to approximate the value of the state at the next timestep. Explicit methods are described by the number of "stages" they contain (number of function evaluations required per timestep), and by their "order" p, which is a measure of the local truncation error caused by each timestep h, and matches a local Taylor series expansion to within an error of $O(h^{p+1})$ [4]. Butcher developed a framework for classifying Runge-Kutta methods based upon their order and number of stages, organizing their coefficients and stage separations graphically in a Butcher Tableau [5]. Simple explicit methods may utilize fixed timesteps, where the forward prediction timestep is the same length each step. More advanced algorithms combine a lower order and a higher order algorithm pair ("embedded algorithms"), using complementary methods such that the function evaluation points may be shared between the two algorithms. The benefit of this is that the difference in the predicted state between the high-order and low-order method provides an estimate of the local solution error, thus allowing the timestep to be adaptively adjusted to the local complexity of the ODE system.

For example, the Dormand-Prince RK5(4) embedded algorithm combines fifth- and fourth-order Explicit Runge-Kutta methods and requires seven stages [6], and the Dormand-Prince RK8(7) embedded algorithm combines eighth- and seventh-order methods and requires 13 stages [7]. The embedded Runge-Kutta stepsize control allows the integration error to be roughly uniformly maintained. However, an initial guess of the stepsize is required to begin the adaptive method. Several algorithms are available for selection of a starting stepsize for a general ODE [8, 9], or, for the case of orbit propagation, heuristics based upon the orbit parameters may be used to select an initial stepsize [10].

Numerical integration of a naturally second-order ODE system (such as the equations of perturbed orbital motion) with an Explicit Runge-Kutta method generally requires decomposition of the system into a set of first-order ODEs (using, for example, the Dormand-Price embedded methods). Naturally second-order Explicit Runge-Kutta methods (double integrator methods) exist that operate directly upon the second-order ODE system. The most widely used are the Runge-Kutta-Nyström methods, for example the RKN12(10) embedded method [11].

Within an explicit method, the state value is predicted using a simple weighted linear combination of evaluations of the ODE function (corresponding to a lower-triangular coeffcient matrix A in the Butcher Tableau), and the intermediate stage values as well as the ultimate future value of the state at the next timestep may be calculated using sequential evaluations of the ODE function. An Implicit Runge-Kutta (IRK) method requires the solution of a set of coupled non-linear equations (corresponding to a general coefficient matrix A in the Butcher Tableau), thus requiring an initial guess of the intermediate stage values in order to begin. The computational cost per step is therefore much greater for an implicit method than an explicit method. However, IRK algorithms have a great advantage over explicit

methods in terms of stability, which makes them ideal for use in stiff ODE systems (systems exhibiting dynamics on both long and short timescales) [4, 5, 12].

A collocation method is a form of numeric solver in which an orthogonal basis set is used to approximate the state values, subject to the proper boundary conditions, with the constraint that the derivative of the approximation exactly satisfies the ODE function at a specifically sampled set of nodes (collocation points). It may be shown that the collocation formulation is a form of Implicit Runge-Kutta, and thus has the same favorable stability properties [3, 13, 14]. Recent work has shown that Collocation IRK methods are able to compete with, or outperform, the state-of-practice numerical methods for perturbed orbit propagation [15–20]. Collocation IRK methods also have the benefit that they are inherently parallelizable numerical integration methods, a claim that the explicit methods cannot make due to their sequential nature.

Multi-step integration methods use the current state values, as well as a set of previous state values called "backpoints", to extrapolate the state at the next timestep. These methods are called predictor-corrector methods because they use the current state and set of backpoints to predict the value of the future state, then add the predicted state to the set of backpoints and use a separate corrector algorithm to refine the approximation of the future state. Integral and derivative operations are iteratively constructed as "difference tables", and the prediction and correction phases are carried out by linearly combining forward, backward, and central differences of the ODE function evaluations along the set of backpoints. If a high-accuracy solution is required, the prediction/correction steps may be iteratively repeated to refine the estimated future state, at the cost of computation time. There are predictor-corrector algorithms to operate on first-order and second-order ODE systems. A widely used method for orbit propagation is a second-order algorithm

called the Gauss-Jackson method (second-sum method), preferred for its stability and truncation properties [10]. In the literature however, the method called Gauss-Jackson is also generally used to refer to the combination of the second-order Gauss-Jackson predictor-corrector, and the first-order Adams predictor-corrector in order to propagate both position and velocity states [21,22]. Typically, predictor-corrector methods utilize fixed length timesteps. However a second-order variable timestep Stormer-Cowell method, with internal error control, was presented by Berry [23]. A serious downside of the predictor-corrector methods is that the initial set of back-points must be initialized by some startup procedure, which can be a computationally costly endeavor requiring many force function evaluations. This downside is not too expensive if a "restart" is rarely required. However, for highly variable non-linearity (a highly eccentricity orbit, for example), one must either use restarts or set a fixed timestep size conservatively small everywhere so as to provide accurate propagation in the most nonlinear areas (near perigee for the case of the highly eccentric orbit). Typically an iterative startup procedure is utilized, beginning from an analytic or semi-analytic approximate orbit solution, or a lower order numerical integrator [21].

Early studies comparing the relative efficiency and achievable accuracy of the various classes of numerical propagators described above, applied to the problem of orbit propagation, have generated the "conventional wisdom" used to make design decisions in legacy software suites. These early studies concluded that for problems where a nearly constant timestep around the orbit may be used (i.e. nearly circular orbits), the multi-step methods are able to deliver the best combination of efficiency and accuracy. For more eccentric orbits the adaptive step-size and inbuilt error estimation of the single-step (Runge-Kutta) methods makes them preferred. Similarly, for extremely non-linear regimes, such as orbital re-entry subject to extreme atmospheric drag perturbations, a high-order single-step method is able to

deliver accuracy that cannot be matched by fixed stepsize multi-step methods. The naturally second-order (double integrator) methods, such as Runge-Kutta-Nystrom, are able to deliver arbitrarily high accuracy solutions with reduced computational cost compared to their single-integrator cousins. However, these methods are unsuitable when the perturbing accelerations involve velocity level variables [10, 24, 25]. These early studies did not include the newer generation of Implicit Runge-Kutta algorithms and collocation methods. A more modern set of studies that includes these algorithms shows that they are able to compete with, or outperform, the more traditional algorithms described above in a serial computation environment [18, 26]. Future parallelized implementations of these modern methods will be able to deliver comparable accuracy results to the state of practice numerical methods, with orders of magnitude reduced computational cost. Next generation software suite designers would be wise to make use of the newly emerging, highly efficient and *parallelizable*, integration algorithms that have been published in recent years. This dissertation focuses on Modified Chebyshev Picard Iteration (MCPI), which is a member of this class of next generation, parallelizable, numerical integration methods.

Modified Chebyshev Picard Iteration is an iterative numerical method for solving linear or non-linear ordinary differential equations. It combines the discoveries of two great mathematicians: Emile Picard (Picard iteration) and Rafnuty Chebyshev (Chebyshev polynomials for orthogonal approximation). The decision to make use of orthogonal Chebyshev function approximation and Picard iteration in a simultaneous manner for solving non-linear ordinary differential equations was first proposed by Clenshaw and Norton in 1963 [27]. Later authors including Shaver, Feagin and Nacozy, and Fukushima further refined the Chebyshev-Picard framework, and the parallel computing implications of the method [28–30]. In 2010, Bai's dissertation extended the earlier MCPI works and proved the capability of the method to outper-

form the state of the practice for numerical integration of ODEs [31]. Bai and Junkins applied MCPI to non-linear IVPs and orbit propagation [32], and showed that MCPI can outperform other higher order integrators such as Runge-Kutta-Nystrom 12(10) [26]. Bai and Junkins have also demonstrated using variations of MCPI to: *(i)* efficiently solve Lambert's transfer problem, *(ii)* solve an optimal control trajectory design problem more accurately and efficiently than the Chebyshev pseudospectral method [33], *(iii)* and to evaluate complex three-body station-keeping control problems formulated as a BVP [34]. In 2013, Bani-Younes' dissertation expanded upon the benefits of MCPI, and orthogonal approximation as a whole, for perturbed orbit propagation [35].

In the above developments, MCPI has been shown to be able to compete with, or outperform, the state-of-the-practice numerical integration algorithms for solution of a variety of initial and two point boundary value problems in a serial environment. MCPI has the added benefit that it is an inherently massively parallelizable method, a claim that is not possible for other competing algorithms except for a subset of the implicit algorithms. Massively parallel MCPI has been shown capable of orders of magnitude performance increase over serial implementations. There is a drawback, however. Tuning of MCPI (time segment lengths and degree of Chebyshev approximation) is problem-dependent. A tuning study must frequently precede the realization of the advantages discussed above.

1.3 Overview

This dissertation presents a body of work that serves to enhance the performance and automation of the Modified Chebyshev Picard Iteration numerical integration method. Special attention is paid to enhancements of MCPI when applied to the equations of perturbed orbital motion, and to ODE systems which intrinsically have

associated conserved quantities. The improvements outlined in this document are the result of an effort to promote MCPI from an algorithm to a broadly applicable toolset for general use. Various enhancements are described such that MCPI may be operated in an autonomous fashion, without a user in the loop, and have the method deliver optimal (or nearly optimal) performance. Additionally, MCPI code libraries are presented such that MCPI may be applied to general ODE systems, without requirement of the user creating their own MCPI implementation.

The remainder of Section 1 describes and derives the MCPI method, with separate MCPI implementations designed for first-order and second-order ODE systems. Within this section, an outline of the MCPI equations are presented, such that someone unfamiliar with the method may gain a general understanding. Appendix D presents a rigorous term-by-term derivation of the equations contained within this introductory section, something that has not been previously published.

Section 2 presents a set of improvements to basic MCPI (as presented in Section 1), for increasing the computational performance of perturbed orbit propagation. Numerically propagating high-fidelity orbits requires evaluation of computationally expensive perturbation force models, and generally these force function evaluations account for the vast majority of computation time. Typically, the spherical harmonic gravity series is the most costly to calculate if high precision solutions are required. Because MCPI is an iterative method, and the force functions must be evaluated on every iteration, there are two strategies for reducing total computation time. The first is to decrease the computational cost per iteration, and the second is to decrease the number of iterations. Sections 2.2 and 2.3 deal with reducing cost per iteration, and Sections 2.4 and 2.5 focus on reducing the number of iterations. In Section 2.2, an adaptive spherical harmonic gravity method is presented that automatically chooses the required spherical harmonic series degree and order

based upon the required acceleration precision and the instantaneous orbital radius. For highly eccentric orbits, the number of terms in the gravity expansion may vary from tens of thousands near perigee to fewer than 50 terms near apogee, without accuracy loss. In Section 2.3, a Taylor series method is presented for approximating high-fidelity gravity accelerations using computationally cheaper lower-fidelity function evaluations, either between subsequent Picard iterations, or for neighboring trajectories in the vicinity of a reference trajectory. Due to the fixed point nature of Picard iteration convergence, the later iterations approaching final convergence result in multiple gravity force evaluations very near fixed points in the force field. The small spatial variations of gravity near each node can be captured with very efficient local models. Section 2.4 presents a method for improving the initial state estimate at each function evaluation point, thus reducing the required number of Picard iterations. In the case that the second-order MCPI formulation is not desired to be used for propagating a naturally second-order system, Section 2.5 describes an efficient method for applying the first-order MCPI formulation.

Whereas Section 2 is focused on improving the performance of MCPI for perturbed orbit propagation, Section 3 describes a body of work designed to automate the process. Before this work, the user was required to select appropriate values of MCPI tuning parameters based upon insights regarding the particular orbit being propagated, or alternatively, some iterative adaptation exercise was required. In order to attain near-optimal performance, this process required familiarity with tuning the MCPI algorithm, as well as heuristic insight to choose tuning parameters based upon orbit elements and the complexity of the gravity model. This presented a barrier for widespread use of the method, as well as inviting invalid conclusions regarding comparisons of MCPI with other methods having fewer tuning parameters or existing automated tuning. In Section 3, two tuning parameter sets are generated

and compared (one is generated empirically and the other by using a global sampling based optimization method called a genetic algorithm). A "plug-and-play" module for autonomous selection of a reasonable (near-optimal) set of tuning parameters for MCPI is described for perturbed orbit propagation.

Section 4 presents a constrained MCPI formulation for application to ODE systems with associated conserved quantities. Between Picard iterations, this constrained method applies a small "state restoration" correction at each function evaluation point along the MCPI segment of approximation, prior to the subsequent Picard iteration. A constraint restoration algorithm is derived for both the first-order and second-order MCPI formulations. The benefits are explored with several examples, and it is shown that the constraint restoration method is able to reduce the required number of Picard iterations, increase the segment length over which MCPI is capable of converging, and improve the resulting solution accuracy for the same local convergence threshold.

In Section 5 a set of MCPI implementations is presented that are designed to be broadly applicable to general ODE systems, and deliver reasonable computational performance to achieve specified accuracy, while requiring minimal user insight. Section 5.1 describes these general MCPI libraries, and briefly demonstrates their use. Additionally, in Section 5.2, a parallelized code framework for perturbed orbit propagation with MCPI is presented. The package is designed to be flexible enough to run (in parallel) on any computer, from the most humble multi-core desktop terminal to the most powerful compute cluster. It is anticipated that this framework will be expanded in the future to launch a new era of accelerated orbit conjunction analysis and space catalog maintenance, as well as provide a baseline comparison for future massively parallel MCPI implementations. This framework is demonstrated on a compute cluster, and examples are shown of parallel space catalog propagation and

parallel uncertainty propagation.

1.4 Modified Chebyshev Picard Iteration

1.4.1 MCPI Fundamentals

An Ordinary Differential Equation (ODE)

$$\dot{\boldsymbol{x}} = \frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(t, \boldsymbol{x}(t)), \quad \boldsymbol{x}(t_0) = \boldsymbol{x_0} \tag{1.1}$$

with a state vector $\boldsymbol{x}(t) \in \mathbb{R}^n$, and a given initial condition $\boldsymbol{x_0}$ at initial time t_0, may be rearranged without approximation to obtain the integral equation

$$\boldsymbol{x}(t) = \boldsymbol{x}(t_0) + \int_{t_0}^{t} \boldsymbol{f}(s, \boldsymbol{x}(s))ds \tag{1.2}$$

Picard proved that a sequence of approximate solutions of increasing accuracy $\boldsymbol{x}^i(t),\ i = 1, 2, 3, ...$ to this integral formulation may be recursively generated by

$$\boldsymbol{x}^i(t) = \boldsymbol{x}(t_0) + \int_{t_0}^{t} \boldsymbol{f}(s, \boldsymbol{x}^{i-1}(s))ds \tag{1.3}$$

Picard's Method of Successive Approximations, now called Picard iteration, constitutes a contraction mapping to the true solution $\boldsymbol{x}(t)$ under broadly applicable assumptions [36]. For any smooth, integrable, single-valued function $\boldsymbol{f}(t, \boldsymbol{x}(t))$, the sequence converges over some interval $(t_f - t_0) < \delta$ for all starting $\boldsymbol{x}^0(t)$ within the range $||\boldsymbol{x}^0(t) - \boldsymbol{x}(t)|| < \Delta$, where δ and Δ are finite and can be roughly bounded. The rate of convergence is largely dictated by $|t_f - t_0|$ and $||\frac{d\boldsymbol{f}}{d\boldsymbol{x}}||$, and is approximately a geometric sequence. The conditions under which the Picard sequence of Equation 1.3 theoretically converges to the true solution of Equation 1.1 are quite broad, but are not analytically bounded. Many researchers have attempted to find

least conservative analytic conditions bounding the convergence of Picard iteration, the history of which is detailed Bai's dissertation [31]. Since the publication of Bai's dissertation, Woollands and Junkins have shown that the Picard-Chebyshev domain of convergence may be very significantly expanded (when applied to the problem of orbit propagation) by utilizing orbit regularization methods such as the Kustaanheimo-Stiefel transform, which transforms a gravitational two-body problem into a harmonic oscillator [37, 38].

Classically, Picard's method requires repetitive integration of the integrand function, and thus is not well suited to solving broad classes of ODEs, or problems with complicated integrands, unless a highly efficient method is used to carry out the integrals. An analytic example of Picard iteration for the solution of a simple, scalar ODE is shown in Appendix B.1. Combining Picard iteration with orthogonal function approximation of the integrand along each trajectory yields an extremely powerful, and broadly applicable, iterative numerical integration method.

In the MCPI method, orthogonal Chebyshev polynomials are used as basis functions to approximate the integrand in the Picard integral. Some useful properties of the Chebyshev polynomial orthogonal basis set are given in Appendix C.1. Chebyshev polynomials reside in the domain $\tau = [-1, 1]$, and can be generated recursively as:

$$T_0(\tau) = 1 \tag{1.4a}$$

$$T_1(\tau) = \tau \tag{1.4b}$$

$$T_{k+1}(\tau) = 2\tau T_k(\tau) - T_{k-1}(\tau) \ , \ k > 1 \tag{1.4c}$$

Derivatives and integrals of the Chebyshev polynomials are defined analytically in terms of the polynomials themselves, which allows Picard iteration to analytically

integrate the integrand function. Therefore all derived state variable approximations are easily, and kinematically consistently, represented as a Chebyshev series. The integration property of Chebyshev polynomials states that

$$\int T_n(\tau)d\tau = \frac{1}{2}\left(\frac{T_{n+1}(\tau)}{n+1} - \frac{T_{n-1}(\tau)}{n-1}\right) \ , \ n \geq 2 \tag{1.5}$$

Unlike traditional step-by-step integrators, MCPI is unique in that long state trajectory arcs are approximated during each Picard iteration. The independent variable in the system dynamics is scaled and shifted such that the timespan of integration is projected onto the domain $(-1 \leq \tau \leq 1)$ of the Chebyshev polynomials, thus the system states can be approximated using the Chebyshev polynomial basis functions. The orthogonal nature of the basis functions means that the coefficients that linearly scale the basis functions can be computed independently as simple ratios of inner products with no matrix inversion.

A key feature of MCPI is the utilization of time nodes with a non-uniform cosine density sampling of the domain of the Chebyshev basis functions, called Chebyshev-Gauss-Lobatto (CGL) nodes:

$$\tau_j = -\cos\left(\frac{\pi}{M}j\right) \ , \ j = 0, 1, 2, .., M \tag{1.6}$$

This sampling scheme is chosen to exploit the discrete orthogonality property of the Chebyshev polynomials of the First Kind. In contrast to the nodes for discrete orthogonality of Legendre polynomials, no iteration of a transcendental equation is required to compute these nodes. As can be seen in Figure 1.1, the CGL sampling has much higher density towards the edges of the ± 1 domain than in the center. This provides, for most approximation problems, a near-uniform approximation error in

spite of the lack of support to the left of $\tau = -1$ and to the right of $\tau = +1$. Of vital importance, the sampling scheme approximately eliminates the Runge phenomena, a common issue in function approximation whereby large approximation errors are created near the segment boundaries due to lack of knowledge of the states outside the boundaries. It can be shown, as a practical matter, that the approximation error amplitudes are much more nearly constant as a consequence of cosine sampling, in comparison to uniform sampling. Furthermore, sampling consistent with the discrete orthogonality conditions of the Chebyshev polynomials (or any other orthogonal polynomial basis functions) eliminates the necessity of a matrix inverse and enables machine precision approximation of the integrand in Picard iteration [39].

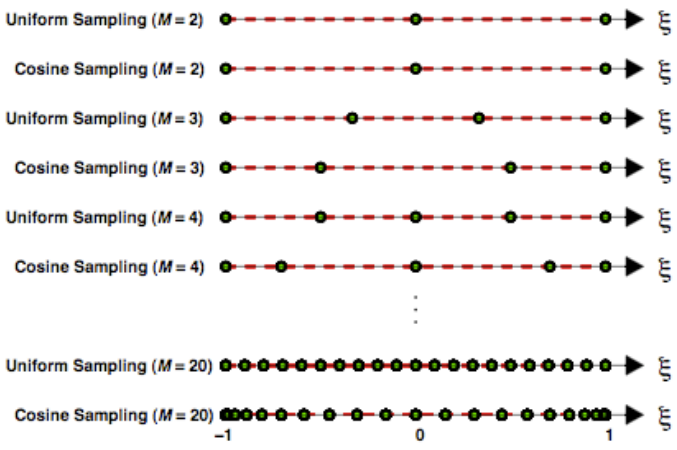


Figure 1.1: Cosine density sampling of the domain of approximation at the Chebyshev-Gauss-Lobatto (CGL) nodes.

Note that any orthogonal polynomial basis function set gives rise to an algorithm analogous to MCPI. However, the Chebyshev polynomials have some advantages: *(i)* the cosine clustering of denser nodes (in order to leverage discrete orthogonality)

results in very small Runge effect (large approximation errors near the ends of the interval), and *(ii)* the nodes for discrete orthogonality are algebraically computed with no need for iteration. Experiments show that the Chebyshev polynomial basis set is slightly preferred over Legendre polynomials for implementation with Picard iteration.

1.4.2 MCPI Derivation

A brief overview of MCPI for first-order Ordinary Differential Equations with state vectors of arbitrary length is presented in this section, for the most part following Bai's derivation, notation, and methodology [26, 31]. Bani-Younes' dissertation presented some minor differences from Bai's original derivation [35]. These differences are noted in this presentation, and a further small deviation from both Bai's and Bani-Younes' work is also noted. These refinements affect the highest degree terms of the approximation, and therefore frequently, but not always, do not significantly affect the numerical results. Appendix D.1.1 is an explicit term-by-term derivation of first-order MCPI that demonstrates all the individually trivial, but collectively important, details leading to the equations presented in this section. This explicit derivation has not previously been published.

Consider a general vector Ordinary Differential Equation (ODE) with a known initial state vector $\boldsymbol{x}(t_0)$ at time t_0, as in Equation 1.1. The objective is to numerically integrate the ODE system to find the state vector $\boldsymbol{x}(t_f)$ at some later time t_f. In practice, time t_f can be arbitrarily long after t_0 (or before t_0). It will be shown later that any segment of interest $(t_f - t_0)$ can be divided into smaller segments over which MCPI is able to efficiently converge, and MCPI may be executed on these shorter segments in a daisy-chain fashion, but for now assume that $(t_f - t_0) < \delta$ is within the convergence domain of a single MCPI segment. As a practical matter, the adaptive

MCPI method described in Section 3.6 guarantees that this assumption is true.

The independent time variable $t_0 \leq t \leq t_f$ for any segment is transformed to a new variable τ, which is defined over the valid domain of the Chebyshev polynomials $-1 \leq \tau \leq 1$, using the transformation

$$t = w_1 + w_2\tau \ , \ w_1 = \frac{t_f + t_0}{2} \ , \ w_2 = \frac{t_f - t_0}{2} \tag{1.7}$$

This transformation is applied to the ODE in Equation 1.1 to transform from $\boldsymbol{f}(t, \boldsymbol{x}(t))$ to a new scaled form of the ODE $\boldsymbol{g}(\tau, \boldsymbol{x}(\tau))$:

$$\boldsymbol{g}(\tau, \boldsymbol{x}(\tau)) \equiv \frac{d\boldsymbol{x}}{d\tau} = \left[\frac{d\boldsymbol{x}}{dt}\right]\left[\frac{dt}{d\tau}\right] \tag{1.8}$$

Substituting the definition of t from Equation 1.7 into Equation 1.8 gives

$$\boldsymbol{g}(\tau, \boldsymbol{x}(\tau)) = w_2\boldsymbol{f}(w_1 + w_2\tau, \boldsymbol{x}), \tag{1.9}$$

and, analogous to Equation 1.3, Picard iteration for this transformed system becomes

$$\boldsymbol{x}^i(\tau) = \boldsymbol{x_0} + \int_{-1}^{\tau} \boldsymbol{g}(s, \boldsymbol{x}^{i-1}(s))ds \quad i = 1, 2, ... \tag{1.10}$$

An initial approximation of the solution $\boldsymbol{x}^0(t)$ is required to begin Picard iteration. MCPI is usually able to converge even with a completely uninformed intial approximation, i.e., setting the solution equal to the initial boundary condition at all nodes. However, the closer the initial estimate is to the true solution, the fewer iterations, and thus fewer function evaluations, will be required to converge. In Section 2.4 several methods for providing an informed initial estimate for perturbed orbit propagation are described.

An N^{th}-order sequence of Chebyshev polynomials are used to approximate the i^{th} Picard estimate of the *system states* on the left hand side of Equation 1.10 as

$$\begin{aligned} \boldsymbol{x}^i(\tau) &\approx \sum_{k=0}^{N}{}'' \boldsymbol{\beta}_k^i T_k(\tau) \\ &\approx \frac{1}{2}\boldsymbol{\beta}_0^i T_0(\tau) + \boldsymbol{\beta}_1^i T_1(\tau) + \boldsymbol{\beta}_2^i T_2(\tau) + ... + \frac{1}{2}\boldsymbol{\beta}_N^i T_N(\tau) \end{aligned} \tag{1.11}$$

where $T_k(\tau)$ is the k^{th} Chebyshev polynomial evaluated at scaled time τ, and $\boldsymbol{\beta}_k^i$ denotes the i^{th} approximation of the k^{th} state coefficient vector. The notation $(\prime\prime)$ in Equation 1.11 will be used ubiquitously, and indicates that the first and the last term in the summation are multiplied by a factor of $\frac{1}{2}$. Each of the $N+1$ $\boldsymbol{\beta}_k^i$ vectors is an $(n \times 1)$ vector, where n is the number of states contained in the system state vector ($\boldsymbol{x}(t) \in \mathbb{R}^n$).

A separate Chebyshev polynomial sequence is used to approximate the $(i-1)^{th}$ Picard estimate of the *integrand* on the right hand side of Equation 1.10 as

$$\begin{aligned} \boldsymbol{g}(s, \boldsymbol{x}^{i-1}(s)) &\approx \sum_{k=0}^{N-1}{}' \boldsymbol{F}_k^{i-1} T_k(s) \\ &\approx \frac{1}{2}\boldsymbol{F}_0^{i-1} T_0(s) + \boldsymbol{F}_1^{i-1} T_1(s) + \boldsymbol{F}_2^{i-1} T_2(s) + ... + \boldsymbol{F}_{N-1}^{i-1} T_{N-1}(s) \end{aligned} \tag{1.12}$$

where $\boldsymbol{F}_k^{i-1}$ denotes the $(i-1)^{th}$ approximation of the k^{th} integrand coefficient vector (each of which is an $n \times 1$ vector, as in Appendix C.1). The notation $(\prime)$ in Equation 1.12 indicates that the first term in the summation is multiplied by a factor of $\frac{1}{2}$. Note that, while an N^{th}-order Chebyshev sequence was used to approximate the system states, an $(N-1)^{th}$-order sequence is used here to approximate the integrand. The reason for this will be shown shortly.

As described in Appendix C.1, there are two closely related methods for orthogo-

nal approximation of a function with a finite number of Chebyshev polynomial basis functions: the least squares Chebyshev approximation method, and the interpolation Chebyshev approximation method. Both methods are valid when sampling the function at the Chebyshev-Gauss-Lobatto (CGL) nodes of Equation 1.6. When the number of CGL sample points M is equal to the Chebyshev order of approximation N, the interpolation method of Equations C.10 is used. When more CGL sample points than the Chebyshev order of approximation are used ($M > N$), the least squares method of Equations C.7 is used. In Equation 1.11 the system states are approximated using an N^{th} order Chebyshev approximation, sampled at $M = N$ CGL nodes, therefore using the interpolation method. In Equation 1.12 the integrand function is approximated using an $(N-1)^{th}$-order Chebyshev sequence, but $M = N$ CGL sample points are still used, therefore using the least squares method.

By evaluating the integrand function on the left-hand-side of Equation 1.12 at the M CGL sample points, the integrand coefficient vectors $\boldsymbol{F}_k^{i-1}$ may be directly solved using the least squares Chebyshev approximation formulation by [40]

$$\boldsymbol{F}_k^{i-1} = \frac{2}{M} \sum_{j=0}^{M} {}'' \boldsymbol{g}(s_j, \boldsymbol{x}^{i-1}(s_j)) T_k(s_j) \tag{1.13}$$

This equation is an inner product of the transformed system dynamics with the k^{th} orthogonal Chebyshev basis function, representing a projection of the true system dynamics (which could have an infinite bandwidth, in theory), onto the finite dimensional basis set. The resulting coefficient vectors $\boldsymbol{F}_k^{i-1}$ are the set that minimize the residual error of the approximation in a least squares sense. Due to orthogonality, the usual normal equations of least squares are trivially invertible, and reduce to the inner products of Equation 1.13 [39, 40]. It is easily demonstrated that even for $(t_f - t_0)$ being a large fraction of an orbit, the finite dimensional approximation of

Equation 1.13 can approach machine precision.

By examination of Equation 1.13, several important truths become clear. The first truth is that each coefficient $\boldsymbol{F}_k$ requires at each timestep s_j the product of the integrand function evaluated at s_j (along the known $(i-1)^{th}$ estimate of the system trajectory) with the k^{th} Chebyshev polynomial evaluated at s_j. However, each of the integrand function evaluations are independent and do not depend upon other timesteps than s_j. This means the integrand evaluations may be performed in any order, or simultaneously in parallel by separate processor threads. The second important truth is that once the integrand function has been evaluated for all timesteps s_j $(j = 0, 1, ..., M)$, the inner products with the k^{th} Chebyshev polynomial to evaluate $\boldsymbol{F}_k$ do no depend upon any other information than T_k and $\boldsymbol{g}(s_j, \boldsymbol{x}^{i-1}(s_j))$. This means that the inner products to solve for $\boldsymbol{F}_0^{i-1}$ through $\boldsymbol{F}_{N-1}^{i-1}$ may also be performed in any order, or simultaneously in parallel. Therefore, there is opportunity for two-fold parallelization in the solution of the integrand coefficients. For a complicated integrand function in which the vast majority of computation time is spent evaluating the accelerations, the equations of perturbed orbital motion for example, the parallelization of the integrand evaluations will greatly reduce computational cost of the overall algorithm. For the case of a trivial integrand function, when the computation time is dominated by the computational overhead of the integration algorithm (the evaluation of these inner products in the case of MCPI), the equations of simple harmonic motion for example, parallelization of the N inner products required to solve for the coefficients has the potential to impact the overall computational cost of the algorithm. A final observation, since the $T_k(\tau)$ are bounded by ± 1, the numerical size of each coefficient $\boldsymbol{F}_k$ represents its maximum contribution. It follows that the final few terms contributing less or more than some tolerance provides convergence insight useful for adaptation of the degree N or the time span of the solution segment.

Substituting the integrand approximation of Equation 1.12 and the state approximation of Equation 1.11 into Equation 1.10, Picard iterations take the form

$$\boldsymbol{x}^i(\tau) = \sum_{k=0}^{N}{}'' \boldsymbol{\beta}_k^i T_k(\tau) = \boldsymbol{x}(-1) + \int_{-1}^{\tau} \left[\sum_{k=0}^{N-1}{}' \boldsymbol{F}_k^{i-1} T_k(s) \right] ds \tag{1.14}$$

where the $\boldsymbol{x_0}$ initial condition term in Equation 1.10 is replaced for notational convenience by $\boldsymbol{x}(-1)$ since the dynamics are scaled to the $[-1, 1]$ domain over which the Chebyshev polynomials are defined. Both sides of Equation 1.14 contain a Chebyshev approximation consisting of coefficients modifying Chebyshev polynomials T_k. The right-hand-side of Equation 1.14 contains the integral of a Chebyshev sequence. Due to the analytic integration property of Chebyshev polynomials (given by Equation 1.5), the right-hand-side projects onto the same basis set, and is itself a Chebyshev sequence. The analytic integration property causes the Chebyshev order to increase by one, thus the reason for approximating the integrand using an $(N-1)^{th}$-order sequence, and the states using an N^{th}-order sequence. This is a slight modification to Bai's original derivation and is first stated in Bani-Younes' dissertation [35]. The integrand coefficients are constants, and are able to be pulled outside the integral, therefore only the Chebyshev functions need to be integrated.

The process of Picard iteration is thus a solution for coefficients modifying like orders of Chebyshev polynomials across the equal sign in Equation 1.14, obtaining the unknown state Chebyshev coefficients $\boldsymbol{\beta}_k^i$ in terms of the known integrand coefficients $\boldsymbol{F}_k^{i-1}$. This process is demonstrated explicitly in Appendix D.1.1. The resulting relationships between the state coefficients and the integrand coefficients,

after imposing the given initial boundary condition, are:

$$\boldsymbol{\beta}_N^i = \frac{1}{N}\boldsymbol{F}_{N-1}^{i-1} \tag{1.15a}$$

$$\boldsymbol{\beta}_{N-1}^i = \frac{1}{2(N-1)}\boldsymbol{F}_{N-2}^{i-1} \tag{1.15b}$$

$$\boldsymbol{\beta}_k^i = \frac{1}{2k}(\boldsymbol{F}_{k-1}^{i-1} - \boldsymbol{F}_{k+1}^{i-1}) \ , \quad k = 1, 2, .., N-2 \tag{1.15c}$$

$$\boldsymbol{\beta}_0^i = 2\boldsymbol{x_0} + 2\sum_{k=1}^{N-1}\left[(-1)^{k+1}\boldsymbol{\beta}_k^i\right] + \left[(-1)^{N+1}\boldsymbol{\beta}_N^i\right] \tag{1.15d}$$

As is first shown by Bani-Younes, Equation 1.15b is not present in Bai's original derivation, and is a direct result of using an $(N-1)^{th}$ order Chebyshev sequence for approximation of the integrand. Additionally, Equations 1.15a and 1.15d are a different form than what is presented by Bai or Bani-Younes, but the origin is documented in Appendix D.1.1. In the frequent circumstance that the highest degree terms have a magnitude of nearly machine zero, the distinction between Bai's formulation and this one is numerically very small.

Having solved for the updated state coefficients, the state trajectory approximation is obtained using Equation 1.11. The updated state trajectory at the i^{th} iteration is fed back into the integrand function during the next Picard iteration. The estimate of the state trajectory is thus iteratively refined until the estimates have converged to within a given stopping criteria. A useful stopping criterion is to require the difference between successive state trajectory estimates $||\boldsymbol{x}^i - \boldsymbol{x}^{i-1}|| < \epsilon$ and $||\boldsymbol{x}^i - \boldsymbol{x}^{i+1}|| < \epsilon$, where ϵ is a user-specified stopping threshold. In the case that some of the states are much larger (or more nonlinear) than others, a normalized stopping criteria $\frac{||\boldsymbol{x}^i - \boldsymbol{x}^{i-1}||}{||\boldsymbol{x}^i||} < \epsilon$ and $\frac{||\boldsymbol{x}^i - \boldsymbol{x}^{i+1}||}{||\boldsymbol{x}^i||} < \epsilon$ is preferred to ensure that the state estimates are of equal fidelity. After convergence, the approximation of the state trajectory is valid over the entire MCPI segment $-1 \leq \tau \leq 1$ (equivalently $t_0 \leq t \leq t_f$).

The values of the states $\boldsymbol{x}(t)$ may be calculated at any time t within the segment by calculating a corresponding τ using Equation 1.7, and then evaluating the Chebyshev polynomials and coefficients at the value of τ using Equation 1.11. The diagram in Figure 1.2 provides a schematic overview of the MCPI steps required to solve an Initial Value Problem.

Frequently the ODE to be solved has some (or all) states known at the initial time $\boldsymbol{x}(t_0) = \boldsymbol{x_0}$ and some (or all) states known at the final time $\boldsymbol{x}(t_f) = \boldsymbol{x_f}$, a construct called a Two Point Boundary Value Problem (TPBVP). Although it was derived for an Initial Value Problem (IVP), MCPI as presented in this section may be used to solve BVPs with only minor modification. Linear boundary conditions at the initial time and at the final time manifest as linear constraints on the state coefficients $\boldsymbol{\beta}^i$. The state coefficient constraints for BVPs are the same as Equations 1.15 for $\boldsymbol{\beta}^i_N$, $\boldsymbol{\beta}^i_{N-1}$, and $\boldsymbol{\beta}^i_k$:

$$\boldsymbol{\beta}^i_N = \frac{1}{N}\boldsymbol{F}^{i-1}_{N-1} \tag{1.16a}$$

$$\boldsymbol{\beta}^i_{N-1} = \frac{1}{2(N-1)}\boldsymbol{F}^{i-1}_{N-2} \tag{1.16b}$$

$$\boldsymbol{\beta}^i_k = \frac{1}{2k}\left(\boldsymbol{F}^{i-1}_{k-1} - \boldsymbol{F}^{i-1}_{k+1}\right), \quad k = 1, 2, ..., N-2 \tag{1.16c}$$

However, the initial and terminal boundary conditions manifest as

$$\boldsymbol{\beta}^i_1 = \frac{(\boldsymbol{x_f} - \boldsymbol{x_0})}{2} - (\boldsymbol{\beta}^i_3 + \boldsymbol{\beta}^i_5 + \boldsymbol{\beta}^i_7 + ...) - \frac{1}{2}\boldsymbol{\beta}_N \tag{1.17a}$$

$$\boldsymbol{\beta}^i_0 = \boldsymbol{x_0} + \boldsymbol{x_f} - 2(\boldsymbol{\beta}^i_2 + \boldsymbol{\beta}^i_4 + \boldsymbol{\beta}^i_6 + ...) \tag{1.17b}$$

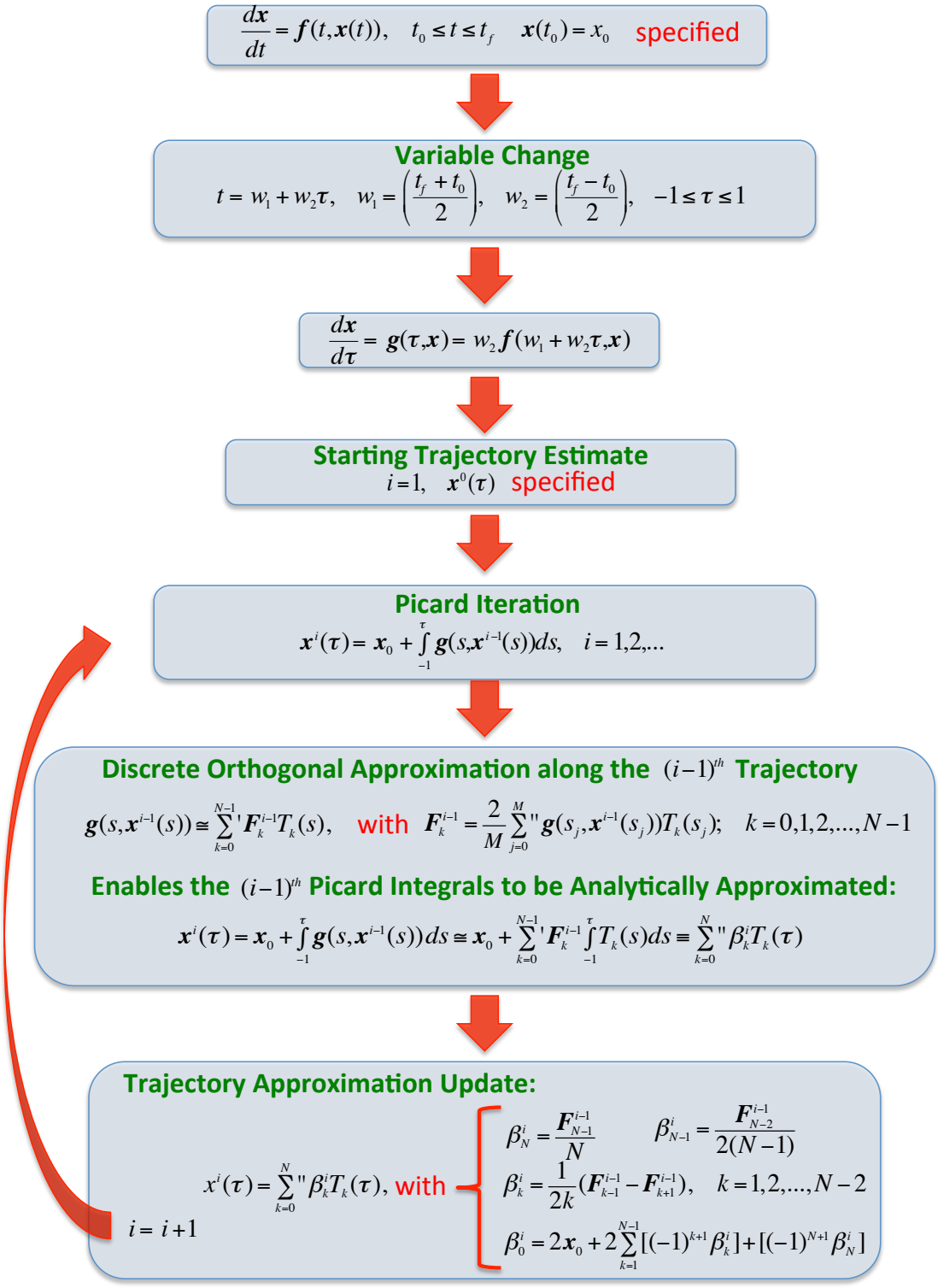


Figure 1.2: Schematic overview of MCPI algorithm for solving an Initial Value Problem (IVP).

for odd N, and as

$$\boldsymbol{\beta}_1^i = \frac{(\boldsymbol{x_f} - \boldsymbol{x_0})}{2} - (\boldsymbol{\beta}_3^i + \boldsymbol{\beta}_5^i + \boldsymbol{\beta}_7^i + ...) \tag{1.18a}$$

$$\boldsymbol{\beta}_0^i = \boldsymbol{x_0} + \boldsymbol{x_f} - 2(\boldsymbol{\beta}_2^i + \boldsymbol{\beta}_4^i + \boldsymbol{\beta}_6^i + ...) - \beta_N \tag{1.18b}$$

for even N.

These coefficient constraints are slightly different than what was given by Bai [31, 33], wherein Equation 1.16b was not present, Equation 1.16a had a slightly different form, and the even/odd forms given by Equations 1.17 and 1.18 was not present. The differences in the highest order terms are again due to approximating the integrand as an $(N-1)^{th}$ order Chebyshev series and the states as an N^{th} order series.

1.4.3 MCPI Vector/Matrix Formulation

For clarity of notation and computational "book-keeping", the MCPI algorithm as presented in Section 1.4.2 is, in practice, restructured into a more compact vector/matrix framework [26]. Computationally this is more efficient than the standard term-by-term formulation in the previous section because it allows a programmer to leverage the vast codebase of vector/matrix algebra libraries that have been increasingly optimized as processor and compiler architectures have evolved. This section presents an overview of the vector/matrix MCPI formulation, and the details are presented in an explicit derivation in Appendix D.1.2.

The i^{th} estimate of the state is written as an $(M+1) \times n$ matrix by stacking the state estimate values for all $(M+1)$ CGL sampled times τ_0 to τ_M as

$$X^i = matrix\{\boldsymbol{x}^i(\tau_0)^T \ ; \ \boldsymbol{x}^i(\tau_1)^T \ ; \ ... \ ; \ \boldsymbol{x}^i(\tau_M)^T\} \tag{1.19}$$

where each $\boldsymbol{x}^i(\tau_j) = [x_1^i(\tau_j) \ , \ x_2^i(\tau_j) \ , \ ... \ , \ x_n^i(\tau_j)]^T$ is an $(n \times 1)$ vector. Equation 1.11,

relating the state estimates $\boldsymbol{x}^i(\tau)$ at a given time τ to the Chebyshev approximation sequence, may be written in vector/matrix form as

$$X^i = \begin{bmatrix} T_0(\tau_0) & T_1(\tau_0) & \dots & T_N(\tau_0) \\ T_0(\tau_1) & T_1(\tau_1) & \dots & T_N(\tau_1) \\ \vdots & \vdots & \dots & \vdots \\ T_0(\tau_M) & T_1(\tau_M) & \dots & T_N(\tau_M) \end{bmatrix} \begin{bmatrix} \frac{1}{2} & & & \\ & 1 & & \\ & & \ddots & \\ & & & \frac{1}{2} \end{bmatrix} \begin{bmatrix} (\boldsymbol{\beta}_0^i)^T \\ \vdots \\ (\boldsymbol{\beta}_k^i)^T \\ \vdots \\ (\boldsymbol{\beta}_N^i)^T \end{bmatrix} \tag{1.20}$$

or, in shorthand

$$X^i = {}^{\beta}T\,{}^{\beta}W\beta^i \tag{1.21}$$

${}^{\beta}W$ is an $(N+1)\times(N+1)$ diagonal weight matrix, and ${}^{\beta}T$ is a $(M+1)\times(N+1)$ matrix of Chebyshev polynomials evaluated at CGL sampled times τ_j. β_i is a $(N+1)\times n$ matrix formed by stacking the components of the individual state coefficients $(\boldsymbol{\beta}_k^i)^T$.

The integrand fit of Equation 1.13 may be written as a vector/matrix system:

$$F^{i-1} = \begin{bmatrix} T_0(\tau_0) & T_0(\tau_1) & \dots & T_0(\tau_M) \\ T_1(\tau_0) & T_1(\tau_1) & \dots & T_1(\tau_M) \\ \vdots & \vdots & \dots & \vdots \\ T_{N-2}(\tau_0) & T_{N-2}(\tau_1) & \dots & T_{N-2}(\tau_M) \\ T_{N-1}(\tau_0) & T_{N-1}(\tau_1) & \dots & T_{N-1}(\tau_M) \end{bmatrix} \begin{bmatrix} \frac{1}{M} & & & & \\ & \frac{2}{M} & & & \\ & & \ddots & & \\ & & & \frac{2}{M} & \\ & & & & \frac{1}{M} \end{bmatrix} \begin{bmatrix} \boldsymbol{G}^{i-1}(\tau_0) \\ \boldsymbol{G}^{i-1}(\tau_1) \\ \vdots \\ \boldsymbol{G}^{i-1}(\tau_{M-1}) \\ \boldsymbol{G}^{i-1}(\tau_M) \end{bmatrix} \tag{1.22}$$

or equivalently,

$$F^{i-1} = {}^{F}T^{T}VG(X^{i-1}) \tag{1.23}$$

F^{i-1} is an $(N\times n)$ matrix of integrand coefficients (n is the length of the state vector: $\boldsymbol{x}(\tau)\in\mathbb{R}^n$), ${}^{F}T$ is an $(M+1)\times N$ matrix of Chebyshev polynomials evaluated at the

CGL sample points ($^{F}T^{T}$ is the transpose of ^{F}T), and V is a $(M+1)\times(M+1)$ weight matrix. $G(X^{i-1})$ is an $(M+1)\times n$ matrix of the values from the evaluations of the integrand approximation function at the $(i-1)^{th}$ estimate of the states $\boldsymbol{g}(\tau_j, \boldsymbol{x}^{i-1}(\tau_j))$

$$\boldsymbol{G}^{i-1}(\tau_j) = \boldsymbol{g}(\tau_j, \boldsymbol{x}^{i-1}(\tau_j))^T \tag{1.24a}$$

$$G(X^{i-1}) = matrix\{\boldsymbol{G}^{i-1}(\tau_0)\ ;\ ...\ ;\ \boldsymbol{G}^{i-1}(\tau_M)\} \tag{1.24b}$$

The state coefficients β^i may be related to the integrand coefficients F^{i-1} by Equations 1.15. These relations may written in vector/matrix form by defining two matrices

$$R = diag\left\{1\ ,\ ...\ ,\ \frac{1}{2r}\ ,\ ...\ ,\ \frac{1}{2(N-1)}\ ,\ \frac{1}{N}\right\}\ ,\ r = 2,3,...,N-2 \tag{1.25}$$

and

$$S = \begin{bmatrix} 1 & -\frac{1}{2} & S(1,3) & \dots & S(1,k) & \dots & S(1,N) \\ 1 & 0 & -1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & -1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \tag{1.26}$$

where the top row of the S matrix has the general term

$$S(1,k) \equiv (-1)^k \left(\frac{1}{k-2} - \frac{1}{k}\right) \tag{1.27}$$

Therefore, Equation 1.15 becomes

$$\beta^i = X_0 + RSF^{i-1} \tag{1.28}$$

where R is an $(N+1) \times (N+1)$ matrix, S is an $(N+1) \times N$ matrix, and X_0 is an $(N+1) \times n$ matrix that enforces the initial boundary condition upon the state coefficients:

$$X_0 = matrix\{2\boldsymbol{x}_\mathbf{0}^T \ ; \ \mathbf{0}_{1xn} \ ; \ ... \ ; \ \mathbf{0}_{1xn}\} \tag{1.29}$$

Summarizing the developments above, the process of Chebyshev approximation and Picard iteration is

$$F^{i-1} = {}^F T^T V G(X^{i-1}) \tag{1.30a}$$

$$\beta^i \quad = X_0 + RSF^{i-1} \tag{1.30b}$$

$$X^i \quad = {}^\beta T^\beta W \beta^i \tag{1.30c}$$

Defining two constant matrices

$$C_\alpha = RS^F T^T V \tag{1.31a}$$

$$C_x = {}^\beta T^\beta W \tag{1.31b}$$

we may simplify the MCPI approximation and update steps to

$$X^i = C_x C_\alpha G(X^{i-1}) + C_x X_0 \tag{1.32}$$

Note that the matrices C_x and C_α, as well as their matrix product $C_x C_\alpha$, are constant once the order of approximation N, and therefore the number of CGL

sample points M, are set. Additionally, for each segment of MCPI, the matrix $C_x X_0$ is constant. These matrices may therefore be generated prior to propagating with MCPI, and do not cause any computational overhead during the process of numerical integration or Picard iteration. Of significance, Bai [31] and Bai and Junkins [26], studied the remarkable behavior of the eigenvalues of $C_x C_\alpha$, and, for the linear case, established rigorous bounds on the maximum time interval over which MCPI convergence is guaranteed.

With the above MCPI vector/matrix formulation, a Picard iteration consists of evaluation of the integrand function at the current state approximation $G(X^{i-1})$, one matrix multiplication, and one matrix addition. Figure 1.3 shows a schematic overview of program flow for solving an IVP using the vector/matrix MCPI formulation.

Boundary Value Problems (BVPs) may be solved using the vector/matrix formulation with only minor changes. Implicitly in the equations above, the S matrix is S_i, i.e. for solution of Initial Value Problems. Similarly, the initial condition constraint matrix X_0 is for enforcing a state constraint at the initial time. Vector/matrix MCPI makes use of an X_{if} matrix and an S_{if} matrix to account for the state coefficient constraints at the beginning and end of the interval of approximation in the BVP case. Similarly, if some states are defined only at the end of the interval we use X_f and S_f. Derivation of the BVP matrices is given by Bai [31, 33]. While the modifications of the MCPI recursion matrices appear small, the impact of the changes on the maximum time interval for convergence is not. For both linear and nonlinear problems, the BVP maximum time interval of convergence of Picard iterations has been found to be about 1/10 of the maximum interval for the corresponding IVP Picard iterations.

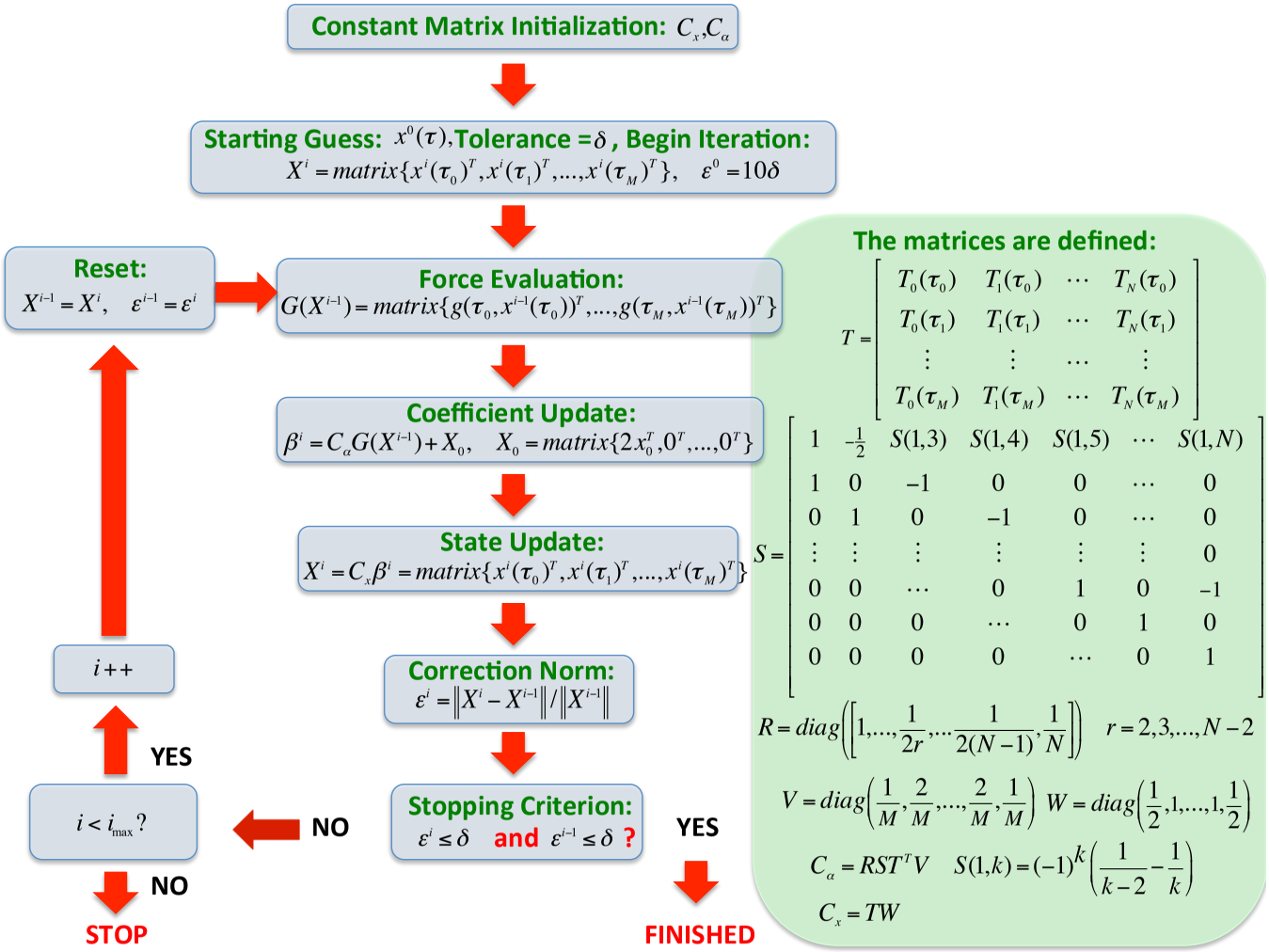


Figure 1.3: Schematic overview of MCPI vector/matrix formulation for solving an Initial Value Problem (IVP).

1.4.4 Second Order MCPI

Consider a second order Ordinary Differential Equation

$$\ddot{\boldsymbol{x}} = \frac{d^2\boldsymbol{x}}{dt^2} = \boldsymbol{f}(t, \boldsymbol{x}, \dot{\boldsymbol{x}}), \quad \boldsymbol{x}(t_0) = \boldsymbol{x_0}, \quad \dot{\boldsymbol{x}}(t_0) = \boldsymbol{v_0} \tag{1.33}$$

with a state vector $\boldsymbol{x}(t) \in \mathbb{R}^n$, a position initial condition $\boldsymbol{x_0}$, and a velocity initial condition $\dot{\boldsymbol{x}}(t_0) = \boldsymbol{v_0}$. The equations of orbital motion are an example of just such a system. The above system could be solved by constructing an augmented state vector $\boldsymbol{z} \in \mathbb{R}^{2n}$ as $\boldsymbol{z} = [\boldsymbol{x}^T, \dot{\boldsymbol{x}}^T]^T$ and numerically integrating the system using the MCPI formulation in the previous sections. In many settings, converting a system of n second-order differential equations into a $2n$-dimensional first-order state space form can be done without penalty. However, it matters significantly in Picard iteration - the number of Picard iterations for the first-order state space has been found to be about twice that of the "natural" second-order system. Bai developed an MCPI formulation to directly operate on the second order equation and calculate the position state trajectory estimate $\boldsymbol{x}(t)$, thus avoiding demotion to a system of first order equations. This is akin to a "double-integrator" numerical method. She called this "position-only MCPI", and noted that the velocity state trajectory estimate $\dot{\boldsymbol{x}}(t)$ may be obtained using the analytic derivative properties of the Chebyshev polynomials [26, 31]. Bani-Younes developed a second-order MCPI formulation for this type of equation, based upon Bai's derivation, which calculates both position and velocity state trajectory estimates in a "cascade" fashion [35].

In this section, an outline is presented of yet one more variation of the MCPI cascade method for second order ODE systems. This new method rigorously enforces the kinematic relationship between the acceleration, velocity, and position state Chebyshev coefficients using the analytic integration property of the Cheby-

shev polynomials, and avoids truncating certain small terms in the Chebyshev series as in the previous developments [31, 35]. An explicit term-by-term derivation of the core equations is presented in Appendix D.2.1, and the vector/matrix formulation is rigorously derived in Appendix D.2.2.

In order to satisfy the second order ODE, the kinematic relationships

$$\frac{d\boldsymbol{x}}{dt} = \dot{\boldsymbol{x}}, \quad \frac{d\dot{\boldsymbol{x}}}{dt} = \boldsymbol{f}(t, \boldsymbol{x}(t), \dot{\boldsymbol{x}}(t)) \tag{1.34}$$

must be true at all times. Transforming the independent time variable from $t_0 \leq t \leq t_f$ to a new variable $-1 \leq \tau \leq 1$, as in Equation 1.7, and similarly scaling the dynamics defined by the ODE, the transformed kinematic relationships, written as

$$\frac{d\boldsymbol{x}}{d\tau} = \boldsymbol{v}, \quad \frac{d\boldsymbol{v}}{d\tau} = \boldsymbol{g}(\tau, \boldsymbol{x}(\tau), \boldsymbol{v}(\tau)) \tag{1.35}$$

must be true at all times. Rearranging the above differential equations to integral equations may be performed without approximation. Picard updates to the velocity states are obtained by

$$\boldsymbol{v}^i(\tau) = \boldsymbol{v_0} + \int_{-1}^{\tau} \boldsymbol{g}(s, \boldsymbol{x}^{i-1}(s), \boldsymbol{v}^{i-1}(s))ds, \quad i = 1, 2, ... \tag{1.36}$$

and kinematic updates to the position states are obtained by

$$\boldsymbol{x}^i(\tau) = \boldsymbol{x_0} + \int_{-1}^{\tau} \boldsymbol{v}^i(s)ds \tag{1.37}$$

Notice that Equation 1.37 is not Picard iteration, it is simply an integral of an exact kinematic constraint between $\boldsymbol{v}^i(\tau)$ and $\boldsymbol{x}^i(\tau)$. Whereas the integrand of the Picard iteration of Equation 1.36 contains the position and velocity state history along the

$(i-1)^{th}$ trajectory approximation, Equation 1.37 is simply the integral of the i^{th} velocity history to obtain the i^{th} position history.

Using the interpolation approximation formulation of Equation C.10, an N^{th}-order Chebyshev polynomial sequence is used to approximate the i^{th} Picard estimate of the system position states sampled at $M = N$ CGL nodes

$$\begin{aligned} \boldsymbol{x}^i(\tau) &\approx \sum_{k=0}^{N}{}'' \boldsymbol{\alpha}_k^i T_k(\tau) \\ &\approx \frac{1}{2}\boldsymbol{\alpha}_0^i T_0(\tau) + \boldsymbol{\alpha}_1^i T_1(\tau) + \boldsymbol{\alpha}_2^i T_2(\tau) + ... + \frac{1}{2}\boldsymbol{\alpha}_N^i T_N(\tau) \end{aligned} \tag{1.38}$$

The velocity states are approximated with an $(N-1)^{th}$-order Chebyshev series, again sampled at $M = N$ CGL nodes, using the least squares Chebyshev approximation formulation of Equation C.7

$$\begin{aligned} \boldsymbol{v}^i(\tau) &\approx \sum_{k=0}^{N-1}{}' \boldsymbol{\beta}_k^i T_k(\tau) \\ &\approx \frac{1}{2}\boldsymbol{\beta}_0^i T_0(\tau) + \boldsymbol{\beta}_1^i T_1(\tau) + \boldsymbol{\beta}_2^i T_2(\tau) + ... + \boldsymbol{\beta}_{N-1}^i T_{N-1}(\tau) \end{aligned} \tag{1.39}$$

As discussed in the Section 1.4.2, the summation in Equation D.45 is from $k = 0$ to $N-1$ since the analytic integration property causes the Chebyshev order of approximation to increase by one, thus the reason for using the least square formulation for the velocity states instead of the interpolation formulation.

A separate Chebyshev polynomial sequence of order $(N-2)$, with $M = N$ CGL

samples, is used to approximate the *integrand* on the right hand side of Equation 1.36

$$\begin{aligned} \boldsymbol{g}(s, \boldsymbol{x}^{i-1}(s), \boldsymbol{v}^{i-1}(s)) &\approx {\sum_{k=0}^{N-2}}' \boldsymbol{F}_k^{i-1} T_k(s) \\ &\approx \frac{1}{2}\boldsymbol{F}_0^{i-1} T_0(s) + \boldsymbol{F}_1^{i-1} T_1(s) + \boldsymbol{F}_2^{i-1} T_2(s) + ... + \boldsymbol{F}_{N-2}^{i-1} T_{N-2}(s) \end{aligned} \tag{1.40}$$

again using the least squares approximation formulation.

Combining Equations 1.38 through 1.40, updates to the integrand and state coefficients take the form

$$\boldsymbol{v}^i(\tau) = {\sum_{k=0}^{N-1}}' \boldsymbol{\beta}_k^i T_k(\tau) = \boldsymbol{v}(-1) + \int_{-1}^{\tau} \left[{\sum_{k=0}^{N-2}}' \boldsymbol{F}_k^{i-1} T_k(s) \right] ds \tag{1.41a}$$

$$\boldsymbol{x}^i(\tau) = {\sum_{k=0}^{N}}'' \boldsymbol{\alpha}_k^i T_k(\tau) = \boldsymbol{x}(-1) + \int_{-1}^{\tau} \left[{\sum_{k=0}^{N-1}}' \boldsymbol{\beta}_k^i T_k(s) \right] ds \tag{1.41b}$$

where $\boldsymbol{v}(-1)$ and $\boldsymbol{x}(-1)$ are the initial conditions of the velocity and position states. Equation 1.41a is a Picard iteration, and Equation 1.41b is a kinematic update. Notice that integrating the acceleration approximation of Equation 1.40 naturally increases the order of the Chebyshev series by one to obtain the corresponding velocity approximation, and by two to obtain the corresponding position approximation. The (usually small) highest order terms in Equations 1.41a and 1.41b could be truncated to enforce an ad hoc desire to approximate position, velocity, and acceleration by a degree $N-2$ polynomial, but we avoid these truncations here. All approximation errors arise at the acceleration level, and the velocity and position approximations are constrained to be kinematically consistent.

The integrand Chebyshev approximation to find the coefficient vectors $\boldsymbol{F}_k^{i-1}$ is performed directly by evaluating the integrand function on the left-hand-side of Equation 1.40 at the $M+1$ CGL sample points. The coefficients are solved by

the same method as in the first order MCPI case, by the expression

$$\boldsymbol{F}_k^{i-1} = \frac{2}{M}\sum_{j=0}^{M}{}''\boldsymbol{g}(s_j, \boldsymbol{x}^{i-1}(s_j), \boldsymbol{v}^{i-1}(s_j))T_k(s_j) \tag{1.42}$$

Having calculated the integrand coefficients, the velocity and position state coefficients may be found by solving across the equal sign in Equations 1.41. This process is demonstrated explicitly in Appendix D.2.1, and summarized here. The velocity state coefficients $\boldsymbol{\beta}_k^i$ are related to the integrand coefficients $\boldsymbol{F}_k^{i-1}$ by the expressions:

$$\boldsymbol{\beta}_{N-1}^i = \frac{1}{2(N-1)}\boldsymbol{F}_{N-2}^{i-1} \tag{1.43a}$$

$$\boldsymbol{\beta}_{N-2}^i = \frac{1}{2(N-2)}\boldsymbol{F}_{N-3}^{i-1} \tag{1.43b}$$

$$\boldsymbol{\beta}_k^i = \frac{1}{2k}(\boldsymbol{F}_{k-1}^{i-1} - \boldsymbol{F}_{k+1}^{i-1}) \ , \quad k = 1, 2, .., N-3 \tag{1.43c}$$

$$\boldsymbol{\beta}_0^i = 2\boldsymbol{v_0} + 2\sum_{k=1}^{N-1}\left[(-1)^{k+1}\boldsymbol{\beta}_k^i\right] \tag{1.43d}$$

The position state coefficients $\boldsymbol{\alpha}_k^i$ are related to the velocity state coefficients $\boldsymbol{\beta}_k^i$ by the expressions:

$$\boldsymbol{\alpha}_N^i = \frac{1}{N}\boldsymbol{\beta}_{N-1}^i \tag{1.44a}$$

$$\boldsymbol{\alpha}_{N-1}^i = \frac{1}{2(N-1)}\boldsymbol{\beta}_{N-2}^i \tag{1.44b}$$

$$\boldsymbol{\alpha}_k^i = \frac{1}{2k}(\boldsymbol{\beta}_{k-1}^i - \boldsymbol{\beta}_{k+1}^i) \ , \quad k = 1, 2, .., N-2 \tag{1.44c}$$

$$\boldsymbol{\alpha}_0^i = 2\boldsymbol{x_0} + 2\sum_{k=1}^{N-1}\left[(-1)^{k+1}\boldsymbol{\alpha}_k^i\right] + \left[(-1)^{N+1}\boldsymbol{\alpha}_N^i\right] \tag{1.44d}$$

Note the presence of the factor of 2 in the denominator of Equation 1.43a, which is

not present in Equation 1.44a or in the first-order MCPI derivation in Equation 1.15a. The presence of the factor of 2 is caused by the fact that the least squares Chebyshev approximation formulation is used for both the velocity state coefficients $\boldsymbol{\beta}_k$ and the integrand coefficients $\boldsymbol{F}_k$, as opposed to Equation 1.44a where the position states $\boldsymbol{\alpha}_k$ utilize the interpolation version of the Chebyshev formulation. Similarly, in the first-order MCPI the Chebyshev sequence on the left-hand-side of the equation uses the interpolation formulation, and the sequence on the right-hand-side uses the least squares formulation. A similar phenomenon may be observed in the difference of structure of the zeroth coefficients (the factor of 2 being absent from the last term) in Equations 1.15d, 1.43d, and 1.44d. Essentially, when the Chebyshev sequence on both sides of the equation uses the least squares formulation, the coefficient expressions will take the form of Equations 1.43a and 1.43d, but when one side of the equation uses the least squares formulation and the other side uses the interpolation formulation, the coefficient expressions will be of the form of Equations 1.44a and 1.44d.

As with the first order MCPI formulation, the second order MCPI may be restructured into a vector/matrix framework. A brief overview is given here, and the details are explicitly presented in Appendix D.2.2.

The i^{th} estimate of the position and velocity states are written as a matrix by stacking the state values for all $M+1$ CGL sampled times τ_0 to τ_M as

$$X^i = matrix\{\boldsymbol{x}^i(\tau_0)^T \ ; \ \boldsymbol{x}^i(\tau_1)^T \ ; \ ... \ ; \ \boldsymbol{x}^i(\tau_M)^T\} \tag{1.45a}$$

$$V^i = matrix\{\dot{\boldsymbol{x}}^i(\tau_0)^T \ ; \ \dot{\boldsymbol{x}}^i(\tau_1)^T \ ; \ ... \ ; \ \dot{\boldsymbol{x}}^i(\tau_M)^T\} \tag{1.45b}$$

where each $\boldsymbol{x}^i(\tau_j) = [x_1^i(\tau_j) \ , \ x_2^i(\tau_j) \ , \ ... \ , \ x_n^i(\tau_j)]^T$ and $\dot{\boldsymbol{x}}^i(\tau_j) = [\dot{x}_1^i(\tau_j) \ , \ \dot{x}_2^i(\tau_j) \ , \ ... \ , \ \dot{x}_n^i(\tau_j)]^T$ is an $(n \times 1)$ vector. Equation 1.38 for the Chebyshev interpolation approximation

of the position states may be expressed as a vector/matrix equation by

$$X^i = \begin{bmatrix} T_0(\tau_0) & T_1(\tau_0) & \dots & T_N(\tau_0) \\ T_0(\tau_1) & T_1(\tau_1) & \dots & T_N(\tau_1) \\ \vdots & \vdots & \dots & \vdots \\ T_0(\tau_M) & T_1(\tau_M) & \dots & T_N(\tau_M) \end{bmatrix} \begin{bmatrix} \frac{1}{2} & & & \\ & 1 & & \\ & & \ddots & \\ & & & \frac{1}{2} \end{bmatrix} \begin{bmatrix} (\boldsymbol{\alpha}_0^i)^T \\ \vdots \\ (\boldsymbol{\alpha}_k^i)^T \\ \vdots \\ (\boldsymbol{\alpha}_N^i)^T \end{bmatrix} \tag{1.46}$$

Similarly, Equation 1.39 for the Chebyshev least squares approximation of the velocity states may be expressed as a vector/matrix equation by

$$V^i = \begin{bmatrix} T_0(\tau_0) & T_1(\tau_0) & \dots & T_{N-1}(\tau_0) \\ T_0(\tau_1) & T_1(\tau_1) & \dots & T_{N-1}(\tau_1) \\ \vdots & \vdots & \dots & \vdots \\ T_0(\tau_M) & T_1(\tau_M) & \dots & T_{N-1}(\tau_M) \end{bmatrix} \begin{bmatrix} \frac{1}{2} & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \begin{bmatrix} (\boldsymbol{\beta}_0^i)^T \\ \vdots \\ (\boldsymbol{\beta}_k^i)^T \\ \vdots \\ (\boldsymbol{\beta}_{N-1}^i)^T \end{bmatrix} \tag{1.47}$$

The above two equations may be summarized as

$$X^i = {}^{\alpha}T\,{}^{\alpha}W\,\alpha^i \tag{1.48a}$$

$$V^i = {}^{\beta}T\,{}^{\beta}W\,\beta^i \tag{1.48b}$$

${}^{\alpha}W$ is an $(N+1)\times(N+1)$ diagonal weight matrix, and ${}^{\alpha}T$, a matrix of Chebyshev polynomials evaluated at CGL sampled times τ_j, is of size $(M+1)\times(N+1)$. ${}^{\beta}W$ is an $N \times N$ diagonal weight matrix, and ${}^{\beta}T$ is of size $(M+1)\times N$. Note that ${}^{\alpha}T$ has a $\frac{1}{2}$ as the last term, whereas in ${}^{\beta}T$ the last term is a 1. This is caused by using the Chebyshev interpolation formulation for the position states, but using the

Chebyshev least squares formulation for the velocity states.

As with the first-order method, the integrand of the ODE is fit using the least squares Chebyshev approximation of Equation 1.13. A Chebyshev approximation of order $N-2$, with $M = N$ CGL sample points is used. During the i^{th} Picard iteration, the $(N-1) \times n$ matrix of integrand coefficients F^{i-1} contains the individual $n \times 1$ integrand coefficient vectors $\boldsymbol{F}_0^{i-1}$ through $\boldsymbol{F}_{N-2}^{i-1}$. The integrand coefficient fit may be written as a vector/matrix expression of the same form as Equations 1.22, or

$$F^{i-1} = {}^{F}T^{T} V G(X^{i-1}, V^{i-1}) \tag{1.49}$$

for short. In the above equation, ${}^{F}T^{T}$ is an $(N-1) \times (M+1)$ matrix of Chebyshev polynomials T_0 through T_{N-2} evaluated at the $M+1$ CGL sample points, and V is $(M+1) \times (M+1)$ Cheybshev weight matrix as defined in Equations 1.22. $G(X^{i-1}, V^{i-1})$ is the result of evaluation of the integrand functions at the $(i-1)^{th}$ approximation of the system states. The slightly confusing notation for the weight matrix V and the i^{th} velocity state approximation V^i is being used in order to remain consistent with previous MCPI publications.

The Picard iteration to relate the unknown velocity state coefficients β^i to the known integrand coefficients F^{i-1} (as given by Equations 1.43) may be written in matrix/vector form as Equations D.82 and D.83. Similarly, Equations 1.44 relating the unknown position state coefficients α^i to the (now) known velocity state coefficients β^i may be written as Equations D.84 and D.85. Equations D.83 and D.85 may be written more compactly as

$$\beta^i = V_0 + {}^{\beta}R^{\beta} S F^{i-1} \tag{1.50a}$$

$$\alpha^i = X_0 + {}^{\alpha}R^{\alpha} S \beta^i \tag{1.50b}$$

V_0 and X_0 are of size $N \times n$ and $(N+1) \times n$, respectively, and are responsible for enforcing the initial boundary conditions for the velocity and position coefficients. ${}^{\beta}R$ and ${}^{\alpha}R$ are square matrices of size $N \times N$ and $(N+1) \times (N+1)$, respectively, and are defined by

$$ {}^{\beta}R = diag\left\{1\ ,\ \ldots\ ,\ \frac{1}{2r}\ ,\ \ldots\ ,\ \frac{1}{2(N-2)}\ ,\ \frac{1}{2(N-1)}\right\}\ ,\ r = 2,3,...,N-2 \quad (1.51a) $$

$$ {}^{\alpha}R = diag\left\{1\ ,\ \ldots\ ,\ \frac{1}{2r}\ ,\ \ldots\ ,\ \frac{1}{2(N-1)}\ ,\ \frac{1}{N}\right\}\ ,\ r = 2,3,...,N-1 \quad (1.51b) $$

These two weight matrices reflect the difference between using the least squares Chebyshev formulation for both sets of states (as with β^i and F^{i-1}), and using the least squares Chebyshev formulation for one set and the interpolation formulation for the other other (as in the the case of α^i and β^i), as discussed above. ${}^{\beta}S$ and ${}^{\alpha}S$ are of size $N \times (N-1)$ and $(N+1) \times N$, respectively, and are the same as the S matrix in first-order MCPI, as defined in Equations 1.26 and 1.27.

Summarizing the expressions so far, we have

$$ F^{i-1} = {}^{F}T^{T} V G(X^{i-1}, V^{i-1}) \quad (1.52a) $$

$$ \beta^i = V_0 + {}^{\beta}R\,{}^{\beta}S F^{i-1} \quad (1.52b) $$

$$ \alpha^i = X_0 + {}^{\alpha}R\,{}^{\alpha}S \beta^i \quad (1.52c) $$

$$ V^i = {}^{\beta}T\,{}^{\beta}W \beta^i \quad (1.52d) $$

$$ X^i = {}^{\alpha}T\,{}^{\alpha}W \alpha^i \quad (1.52e) $$

Defining constant matrices

$$C_\alpha \equiv RS^F T^T V \tag{1.53a}$$

$$C_x \equiv TW \tag{1.53b}$$

$$C_\gamma \equiv RS \tag{1.53c}$$

a Picard update is of the form

$$F^{i-1} = {}^F T^T V G(X^{i-1}, V^{i-1}) \tag{1.54a}$$

$$\beta^i = V_0 + {}^\beta C_\alpha G^{i-1} \tag{1.54b}$$

$$\alpha^i = X_0 + {}^\alpha C_\gamma \beta^i \tag{1.54c}$$

$$V^i = {}^v C_x \beta^i \tag{1.54d}$$

$$X^i = {}^x C_x \alpha^i \tag{1.54e}$$

Note that the matrices C_x, C_γ, and C_α, as well as their matrix products, are constant once the order of approximation N, and therefore the number of CGL sample points M, are set. These matrices may therefore be generated once prior to propagating with MCPI, and do not cause any computational overhead during the process of Picard iteration.

2. IMPROVEMENTS TO MCPI FOR ORBIT PROPAGATION

2.1 Introduction

Perturbed orbital motion is governed by a second order, non-linear, vector differential equation

$$\ddot{\boldsymbol{r}} = -\frac{\mu}{r^3}\boldsymbol{r} + \boldsymbol{a_d} \tag{2.1}$$

In this equation, $\boldsymbol{r}$ is the instantaneous position vector $\boldsymbol{r} = [x, y, z]^T$, r is the magnitude of the position vector, and μ is called the standard gravitational parameter. For an object of mass m around the Earth (with mass M), μ is defined by

$$\mu = G(M + m) \approx GM \approx 398600.4418(9)\ km^3/s^2 \tag{2.2}$$

since $M >> m$ for any current man-made orbital object. The $\boldsymbol{a_d}$ term is the contribution to the instantaneous acceleration due to disturbance accelerations, called perturbations. If $\boldsymbol{a_d} = \boldsymbol{0}$, the equation describes unperturbed Keplerian motion. The equations of Keplerian motion have an analytic solution, called the Lagrange/Gibbs F and G solution [41]. More realistically, the $\boldsymbol{a_d}$ term contains disturbance accelerations due to the gravitational potential of the non-spherical Earth, drag acceleration due to the atmosphere, third-body gravitational accelerations due to the sun and moon, radiation pressure from sunlight, and arbitrarily many other complex forces due to the physics of motion in the space environment, as well as thrust induced acceleration. The generally perturbed equations of motion do not have an analytic solution, and must be solved by numerical integration. By far, the most computationally expensive part of numerically solving the orbital motion equations is the evaluation of the complex disturbance accelerations $\boldsymbol{a_d}(\boldsymbol{r}, \dot{\boldsymbol{r}}, t)$. Typically all of the

overhead computation time due to the core processes of a numerical integrator are a small fraction of the total computation time, dwarfed by the force function evaluations.

This section describes several algorithmic improvements to the basic MCPI algorithm that increase the performance when integrating the equations of perturbed orbital motion. Virtually all of these enhancements center on methods to "legally cheat" by drastically reducing the cost of local force function computations *without introducing significant approximation errors*. Aside from the Cascade method MCPI in Section 2.5, these techniques are applicable to broad classes of numerical integration algorithms.

2.2 Radially Adaptive Gravity

Because the Earth is not a perfect sphere with uniform mass distribution, the gravitational potential in the vicinity of the Earth is a complicated nonlinear scalar field. In order to compute high precision orbit predictions with a numerical propagator, the local gravitational acceleration must be computed to high precision along the entire orbit trajectory. This local high precision gravity calculation is consistently one of the most computationally expensive aspects of perturbed orbit propagation at state of the art precision.

The gravitation potential field of the Earth at a particular radius, latitude, and longitude (r, ϕ, λ) is modeled as a spherical harmonic series

$$U = \frac{\mu}{r}\left[1 + \sum_{l=2}^{\infty}\sum_{m=0}^{l}\left(\frac{R}{r}\right)^{l} P_{l,m}[\sin(\phi)]\left\{C_{l,m}\cos(m\lambda) + S_{l,m}\sin(m\lambda)\right\}\right] \tag{2.3}$$

where $C_{l,m}, S_{l,m}$ are empirically determined coefficients, R is the radius of the Earth, and $P_{l,m}$ are recursively defined associated Legendre functions [42]. The $C_{l,m}$ and

$S_{l,m}$ are in fact higher order mass moments that could be computed exactly if the true density distribution and geometry of the Earth were known. These have been "learned" by solving the inverse problem given almost six decades of tracking actual satellite orbits. Implemented as a computational subroutine, Equation 2.3 and its gradient are a double `for` loop over the degree l and order m of the gravity field. The max degree and order are generally chosen by the user/analyst depending upon the orbital regime, the required accuracy of the solution, the available computational resources, or other heuristic criteria. The equation above is for a "square" gravity model, meaning $l_{max} = m_{max}$.

The normalized $\left(\frac{R}{r}\right)^l$ factor in the summation of Equation 2.3 causes the higher order terms in the series to very quickly decrease in magnitude as the radial distance from the Earth increases. The coefficients $C_{l,m}$ and $S_{l,m}$ multiplying the sine and cosine terms roughly decrease in magnitude with increasing degree and order (roughly but not strictly, nor monotonically). The size of the $C_{l,m}$ and $S_{l,m}$ coefficients reduce from about 10^{-3} for $l_{max} = 2$ to about 10^{-6} for $l_{max} \geq 50$. The combined effect of these factors is that the potential field has many contributing high order terms at low altitude, but very quickly becomes dominated by a few low order terms as the radius increases.

We mention that l_{max} exceeding 200 is required if we wish to compute "atmosphere skimming" low altitude orbits with state of the art precision. Therefore the gravitational potential series in Equation 2.3 and the three vector components of the gradient can result in tens of thousands of terms to be computed. For this reason, many previous researchers have developed methods to decrease the computational burden of gravitational field calculations. These methods generally surrender the elegance of the global spherical harmonic series in favor of local approximation of the higher degree and order terms. Junkins developed an *a priori* finite element method

gravity representation, designed for in-flight applications, which allowed the local gravitational perturbation potential to be modeled before flight using a lower order 3D orthogonal basis set, greatly reducing the in-flight computational cost [43–45]. Bani-Younes and Junkins revisited this work using orthogonal Chebyshev polynomials and modernized it for recent processor developments and parallel computing architectures [35, 39]. These methods, while greatly reducing the necessary real-time computations required in order to calculate a high fidelity gravity acceleration, do so at the cost of added memory requirements (to store the local FEM coefficients). Looking at the problem from a different direction, Vallado quantified the output trajectory error as a result of truncating the gravity series at different degree and order, and drew some conclusions about the required degree and order to retain in order to achieve certain trajectory propagation errors [46]. In this section we present a method, conceptually related to Vallado's approach, to decrease the real-time computational cost of high-precision gravity approximation.

The gravitational acceleration is computed with a gradient of the potential field

$$\boldsymbol{a} = \boldsymbol{\nabla} U \tag{2.4a}$$

$$\boldsymbol{a} = \left[\frac{\partial U}{\partial r}\right]\left[\frac{\partial r}{\partial \boldsymbol{r}}\right]^T + \left[\frac{\partial U}{\partial \phi}\right]\left[\frac{\partial \phi}{\partial \boldsymbol{r}}\right]^T + \left[\frac{\partial U}{\partial \lambda}\right]\left[\frac{\partial \lambda}{\partial \boldsymbol{r}}\right]^T \tag{2.4b}$$

where each of the partial derivative terms of the potential $\frac{\partial U}{\partial (\cdot)}$ is also a double summation with the same ingredients as the potential series of Equation 2.3. For instance, the partial derivative of the potential with respect to the radius is

$$\left[\frac{\partial U}{\partial r}\right] = -\frac{\mu}{r^2}\left[1 + \sum_{l=2}^{\infty}\sum_{m=0}^{l}\left(\frac{R}{r}\right)^l (l+1) P_{l,m}[\sin(\phi)]\{C_{l,m}\cos(m\lambda) + S_{l,m}\sin(m\lambda)\}\right] \tag{2.5}$$

The acceleration exhibits similar radial trends with respect to the number of

relevant terms as the potential field. This is illustrated in Figure 2.1, which shows contours of the local variations of the radial acceleration. This is calculated using the EGM2008 spherical harmonic gravity model with maximum degree and order 200 [47]. Near the surface of the Earth the acceleration perturbations are largest, and vary spatially quite rapidly, because of the large number of non-negligible terms in the series of Equation 2.3. However, at a radial distance of 3 Earth radii the acceleration spatially varies much more smoothly and is two orders of magnitude smaller.

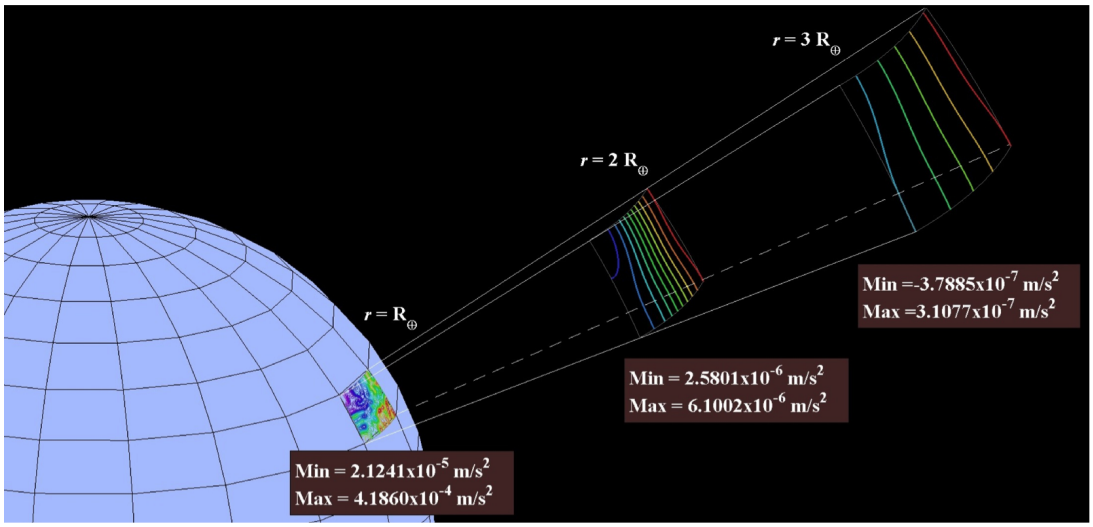
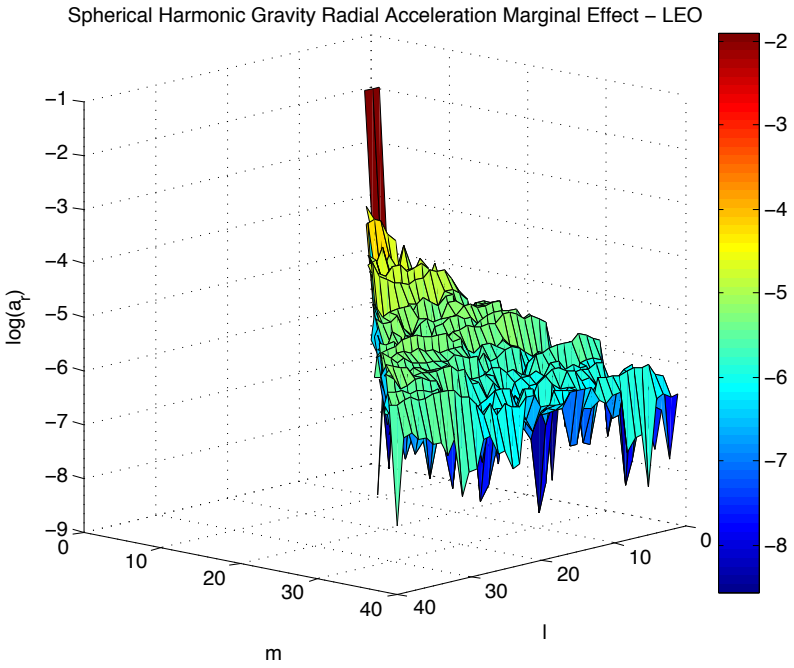


Figure 2.1: Local variations of the gravitational acceleration at various radii.

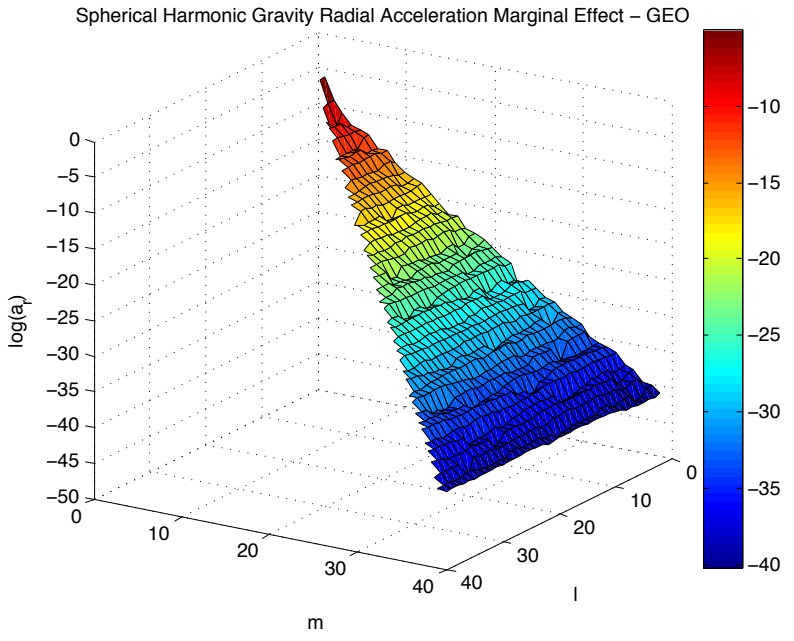
This trend may be examined more rigorously by plotting the individual contribution from each of the terms in the double summation of the acceleration, as shown in Figure 2.2. These plots show the marginal contribution to the sum total of the radial acceleration due to each individual term in the double summation, up to $l_{max} = m_{max} = 40$. Figure 2.2a is for an equatorial point in Low Earth Orbit,

330km altitude in this case, where $\left(\frac{R}{r}\right)^l \approx \left(\frac{1}{1.052}\right)^l$. Figure 2.2b is for a point in the Geostationary belt at an altitude of 35,786km, where $\left(\frac{R}{r}\right)^l \approx \left(\frac{1}{6.61}\right)^l$. For the GEO case, the inverse powers of radii very quickly cause the marginal contributions of any higher order terms to be negligible. Conversely, at the LEO radius the inverse powers are too weak to cancel out the rest of the terms, and all of the plotted terms are significant to within the limits of double precision algebra. The point of all this is, if an analyst decides that an acceleration precision of 8 significant figures is sufficient for his or her solution, the required degree and order to achieve that is vastly different for the two cases. More importantly, as a function only of radius, the l_{max} to achieve any prescribed tolerance can be known in advance, with certainty, and without any real time computation. In GEO, this analyst can "get away with" using a 6^{th} order gravity series and be confident that they have more than sufficient precision for double precision acceleration. In LEO, the acceleration precision will achieve 7 significant figures if the gravity series is arbitrarily truncated at 40^{th} order. In fact, the analyst needs to use a much higher order series if (say) a 9 digit accurate acceleration is desired.

Current practice for perturbed orbit propagation is to select a conservative value of maximum degree and order such that at perigee the dynamics are modeled with sufficient precision. Then, the same maximum degree and order is used to evaluate the gravitational acceleration at all points along the orbit. This is reasonable for a nearly circular orbit, where all points along the orbit are roughly the same altitude. It is computationally wasteful, however, for an eccentric orbit. Consider the example of a GEO Transfer Orbit which has a perigee altitude in LEO, and an apogee near GEO. In this case, as we have just seen, we need a much higher order series ($\approx 200^{th}$ order if double precision accuracy is desired) to accurately predict the dynamics near perigee, but could calculate the same fidelity acceleration using a 6^{th} order series near



(a) LEO Orbit.



(b) GEO Orbit.

Figure 2.2: Marginal radial gravitational acceleration magnitude as a function of spherical harmonic degree and order.

apogee.

Because the equation of gravitational acceleration contains a double summation, the computational cost of evaluating a series of maximum degree and order L varies as

$$Cost_L \approx CL^2 + \epsilon \tag{2.6}$$

where C is a constant that depends upon the details of the particular implementation (programming language, programmer skill, compiler, processor architecture, etc), and ϵ is a small startup overhead time. Figure 2.3 shows the roughly quadratic trend in the averaged computational runtime of 20 evaluations of the Matlab built-in EGM2008 spherical harmonic gravity function for various maximum degree and order. Using Equation 2.6, the relative computational cost of a gravity model of order L compared to a model of some higher order L_{max} is proportional to

$$E = \frac{Cost_L}{Cost_{L_{max}}} \propto \left(\frac{L}{L_{max}}\right)^2 \tag{2.7}$$

At an arbitrary point along an eccentric orbit, if the same desired acceleration precision may be achieved with a gravity model of order L as the model of order L_{max} is able to achieve at the orbit perigee, then the *equivalent gravity cost* E at that point is given by the proportionality relation in Equation 2.7.

This motivates the discussion of a *radially adaptive* spherical harmonic gravity model for orbit propagation, first presented in [48]. A desired acceleration precision δ is specified by the user/analyst. At each gravity series function evaluation point (with instantaneous orbital radius r) around the orbit, an L^{th} degree/order gravity series is evaluated, where $L = L(r, \delta)$. The L at each evaluation point is chosen such that $L = max\{l_{max}, m_{max}\} \leq L_{max}$, where l_{max} and m_{max} are the degree and

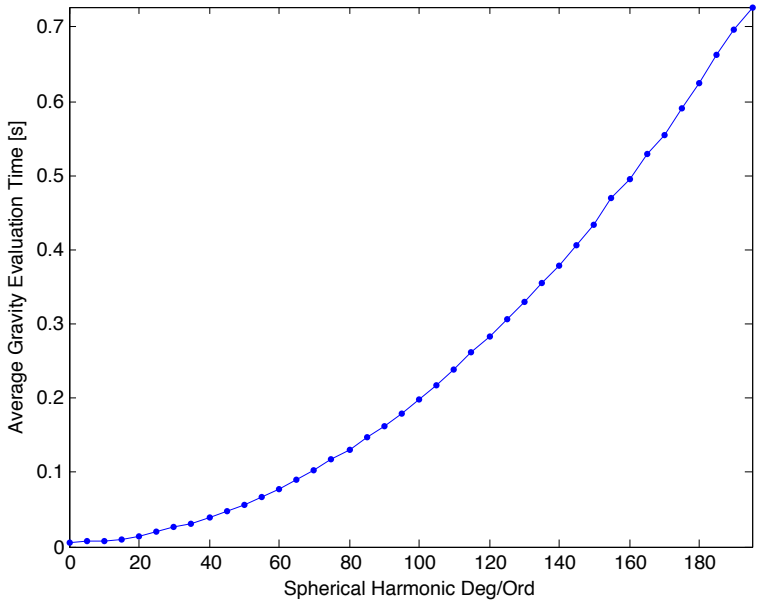


Figure 2.3: Spherical harmonic gravity computational cost as a function of maximum degree/order - calculated empirically using Matlab built-in EGM2008 gravity function.

order of the highest order term in the double summation with magnitude greater than δ, and L_{max} is the heuristically chosen maximum degree and order that would otherwise be used for every function evaluation. This implies no assumption of a monotonically decreasing series. Neither does it cherry-pick higher order harmonic terms while neglecting intermediate order terms, a situation which could potentially systematically eliminate harmonic gravitational perturbations.

An empirical method for a one-time *a priori* determination of the appropriate radially adaptive maximum gravity degree/order $L = L(r, \delta)$ is straightforward to implement. Looping over the individual terms in the double summation of the spherical harmonic gravity series, as shown in Figure 2.2, allows the marginal contribution from each term to be studied. l_{max} and m_{max}, the maximum degree and order terms

which contribute more than tolerance δ, are located in the process. Choosing to maintain a square gravity field for simplicity, we find the maximum required degree and order at a given spatial position (r, ϕ, λ) as $L = max\{l_{max}, m_{max}\} \leq L_{max}$. To ensure that the spatial sampling is representative and comprehensive, 1000 sample points are distributed uniformly around the unit sphere [49], as shown in Figure 2.4. The process of determining the required degree and order is performed at all uniformly spaced sample points on a spherical shell of radius r. Choosing the most conservative value gives $L = L(r, \delta)$. The process is repeated for a range of r values $(r_{min} \leq r \leq r_{max})$ from Low Earth Orbit to above Geostationary Earth Orbit.

A different semi-analytic method for determining the maximum degree/order was presented earlier this year [48]. The idea behind this method is quite elegant[1], and it makes the (offline) computation of the maximum degree and order much faster than the empirical method described above. By examination of Equation 2.3 it is clear that the contribution to the gravitation potential for each term l, m is a product of two factors: the sine and cosine terms scaled by the gravity field coefficients (in the curly brackets), and the associated Legendre function scaled by the inverse powers of the radius. The magnitude of the contribution of the sine and cosine terms is bounded by $\sqrt{C_{l,m}^2 + S_{l,m}^2}$ because the sine and cosine factors themselves are bounded by ± 1. The magnitude of the associated Legendre function scaled by the inverse powers of radius will be at its maximum (for a given radius) at some geocentric latitude. Therefore, the problem is reduced to finding the largest contribution of the product of these two at the single latitude where the Legendre function is the greatest, eliminating the need for evaluation at spherical shells around the Earth. In this manner, it is trivial to determine the maximum degree and order that has terms contributing to the gravitational potential greater than a given threshold. This procedure is slightly more

[1]Credit is due to my colleague Austin Probe for developing this semi-analytic method.

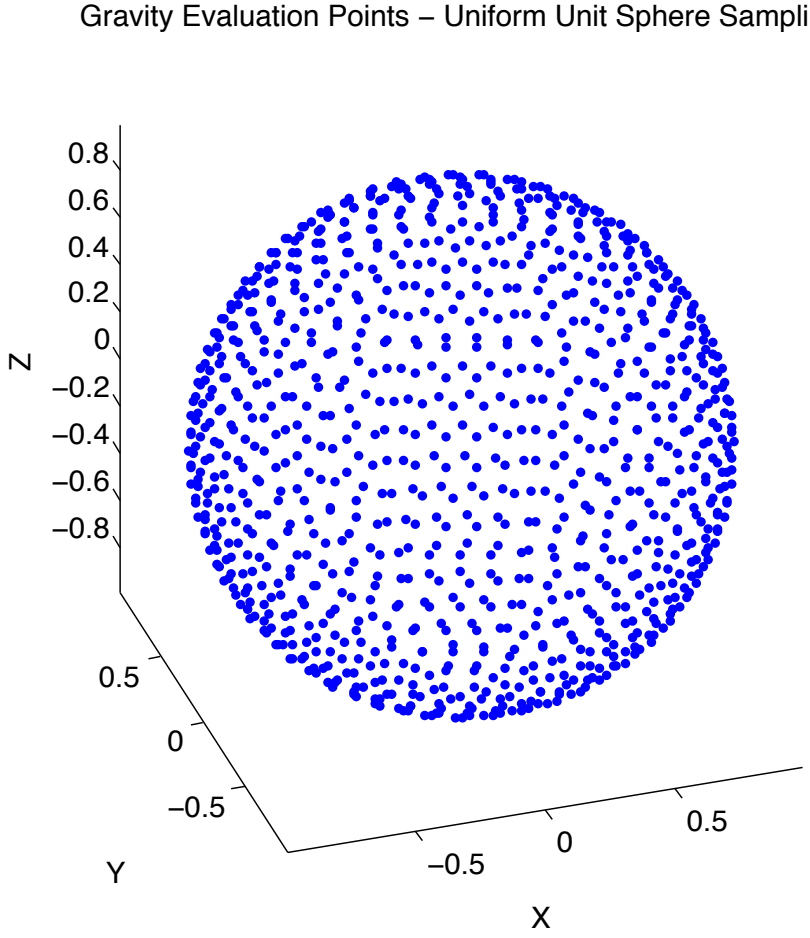


Figure 2.4: Uniformly spaced samples around unit sphere in Earth Centered Earth Fixed (ECEF) coordinates.

complicated for determining contributions to the gravitational acceleration (gradient of the potential), however the same principles may be applied.

Regardless if the empirical method or the more elegant semi-analytic method is used, a cosine-like sampling ξ is defined to determine the radii at which to evaluate the gravity model (as opposed to simple evenly-space sampling):

$$r = r_{min} + (r_{max} - r_{min}) \left[1 - \cos\left(\frac{\pi}{4}(1+\xi)\right)\right] \tag{2.8a}$$

$$\xi = \frac{4}{\pi} \cos^{-1}\left(1 - \frac{(r - r_{min})}{(r_{max} - r_{min})}\right) - 1 \tag{2.8b}$$

where $-1 \leq \xi \leq 1$. This sampling scheme, as shown in Figure 2.5, allows for a concentration of evaluation points near the Earth's surface where the rate of change in the size of the perturbations is the greatest. Additionally, it allows the required degree and order $L = L(r, \delta)$ to be approximated with orthogonal Chebyshev polynomials, using a procedure developed by Junkins and Bani Younes [35, 39].

The result of both the empirical and the semi-analytic method is a two-dimensional look-up table of the required maximum gravity degree and order L, as a function of radius r and desired acceleration tolerance δ. The resulting look-up table (generated by the empirical method) is plotted as a surface in Figure 2.6, which was generated for the EGM2008 spherical harmonic gravity model [47]. The values are truncated at maximum degree and order 100 in this example, but the procedure is valid for arbitrary maximum degree and order, and a similar surface may be generated for any value of L.

Returning to the example of a GEO Transfer Orbit (GTO), the effect of using the radially adaptive gravity method is demonstrated. A GEO Transfer Orbit is an elliptical orbit ($e \approx 0.6$ in this case) with perigee in the Low Earth Orbit region, and apogee at GEO altitude. As an illustration, a single orbital period of this GTO

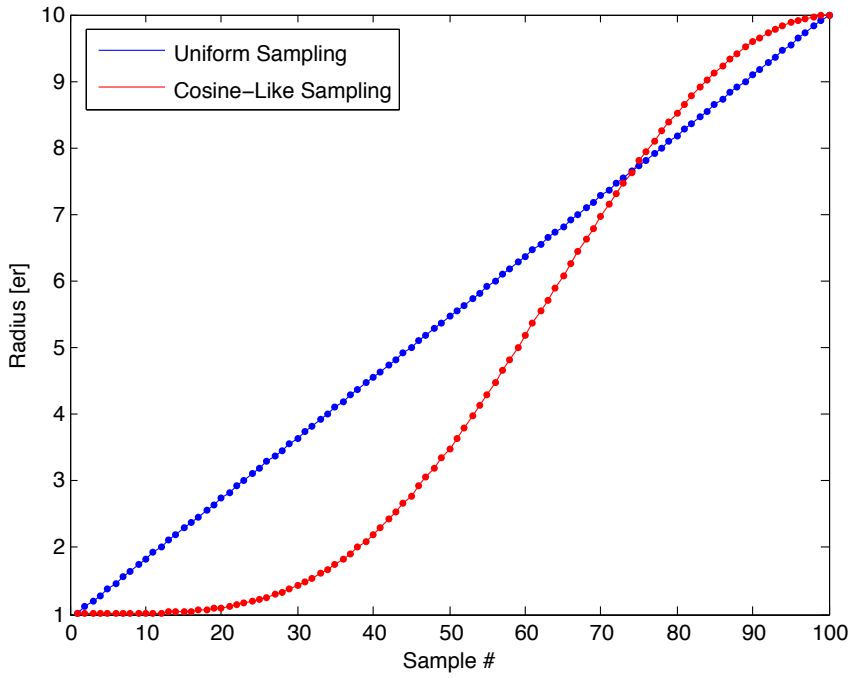


Figure 2.5: Comparison of uniform and cosine-like radial sampling.

orbit is propagated, starting from perigee, using MCPI. Three MCPI segments are used, with segment breaks as determined by the empirical tuning parameter set from Section 3.3, and Chebyshev order of approximation $N = 40$. The top plot of Figure 2.7 shows the instantaneous orbital altitude (above the surface of the Earth) around the orbit, plotted with respect to the orbit period. The bottom plot in Figure 2.7 shows the radially adaptive spherical harmonic maximum degree and order L, using a tolerance of $\delta = 10^{-15}$. The maximum allowable (in this example) degree and order $L_{max} = 40$. Near perigee (far left and far right), the algorithm chooses to use $L = L_{max} = 40$. However, as the radius increases, the required maximum degree and order drops to about $L = 10$ near apogee. Looking back to Equation 2.6, a reduction of required gravity order from 40 to 10 represents a roughly $\left(\frac{10}{40}\right)^2 = 1/16$ equivalent gravity cost at each function evaluation point near apogee.

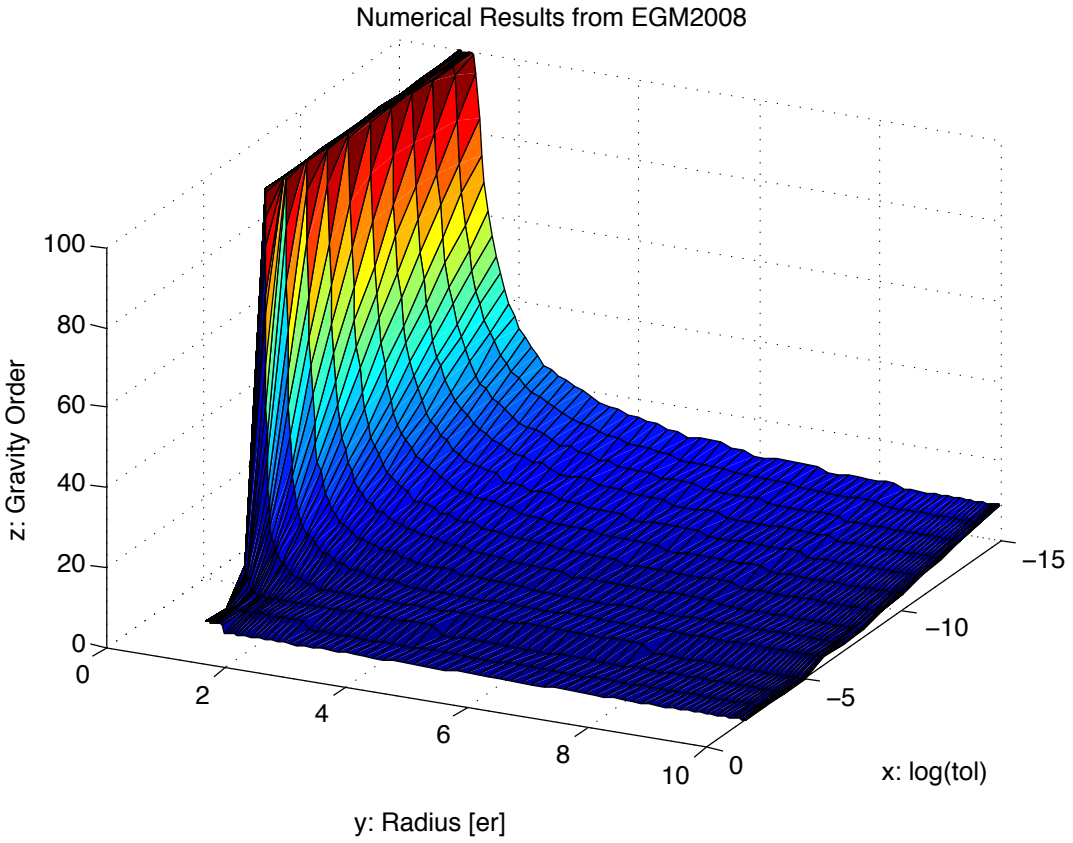


Figure 2.6: Required spherical harmonic gravity degree and order as a function of radius and desired acceleration tolerance.

For propagation of one orbital period, and assuming three MCPI segments of order 40, each iteration of the reference trajectory in which a full gravity evaluation is performed (the meaning of this is explained in Section 2.3) has an equivalent gravity cost of 123 (one full gravity evaluation per node, with no radial adaptation). Using the radially adaptive gravity model, each iteration in which a full gravity evaluation is performed has an equivalent gravity cost of 48.52, or more than 2.5X computational savings. Should we use an L_{max} of 200, the savings is approximately one order of magnitude.

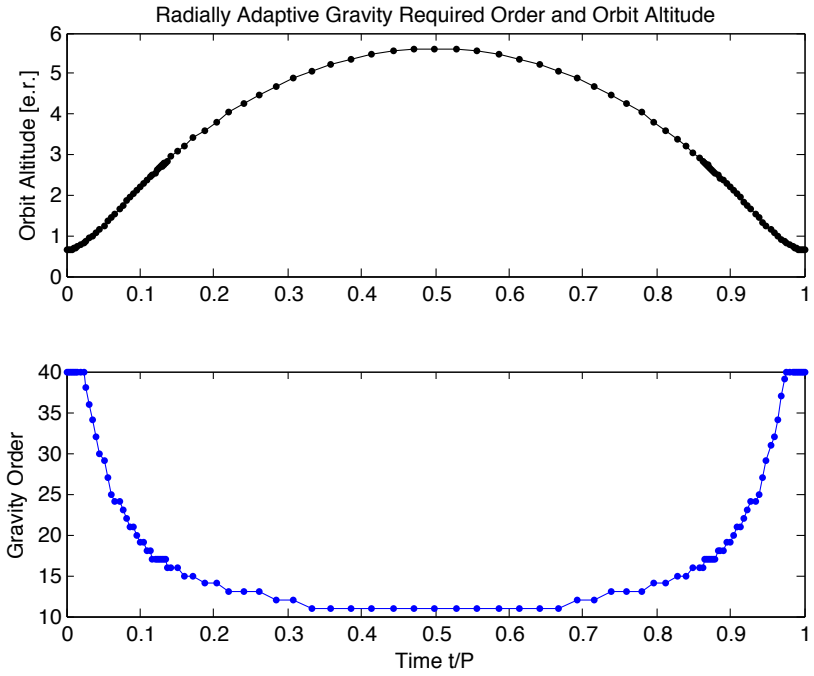


Figure 2.7: Radially adaptive gravity method - instantaneous orbital altitude and required maximum degree and order L along GEO transfer orbit.

The top plot of Figure 2.8 shows the propagated position states $\boldsymbol{r} = [x, y, z]^T$ around the GTO orbit. The effect of the cosine sampling of the time domain at the

CGL nodes (as described by Equation 1.6 and Figure 1.1) of each segment causes the relative sample density near perigee at the endpoints, and the relative sparsity near apogee at the center. The bottom plot of Figure 2.8 shows the achieved Hamiltonian preservation from MCPI propagation without radial adaptation (reference trajectory - red curve), and with radial adaptation (black curve). The differences are hardly observable to plotting precision, and both algorithms are able to achieve relative Hamiltonian preservation on the order of 10^{-14}.

The corresponding difference in the output trajectory between using MCPI with the radially adaptive gravity, and without, are shown in Figure 2.9. The top plot is the deviation in position states, normalized by the magnitude of the position states ($|\boldsymbol{r}_{ref} - \boldsymbol{r}_{adaptive}|/|\boldsymbol{r}_{ref}$), and the bottom plot is the normalized deviation in velocity states ($|\boldsymbol{v}_{ref} - \boldsymbol{v}_{adaptive}|/|\boldsymbol{v}_{ref}$). To numerical precision, the position and velocity are identical until the trajectory re-approaches perigee, where a slight deviation occurs. However, this deviation is nearly a machine precision error of $\delta = 10^{-15}$ (less than one order of magnitude greater than the achievable limit of double precision: $\backsim 16$ decimal places). These deviations are comparable to the errors in the reference trajectory itself. The conclusion: radial adaptation, for eccentric orbits, leads to dramatic speedup with no accuracy loss.

2.3 Taylor Series Gravity Model

The radially adaptive method described in the previous section is a simple, elegant means for reducing the cost of computing accelerations from the global gravity model. A multiple fidelity approach to evaluating force functions can greatly reduce computational effort in an iterative numerical algorithm such as MCPI or Implicit Runge-Kutta (IRK) [15–17, 50]. The motivation for this is the fact that, during the initial few iterations (when the nodal convergence errors have 10^{-4} or greater relative

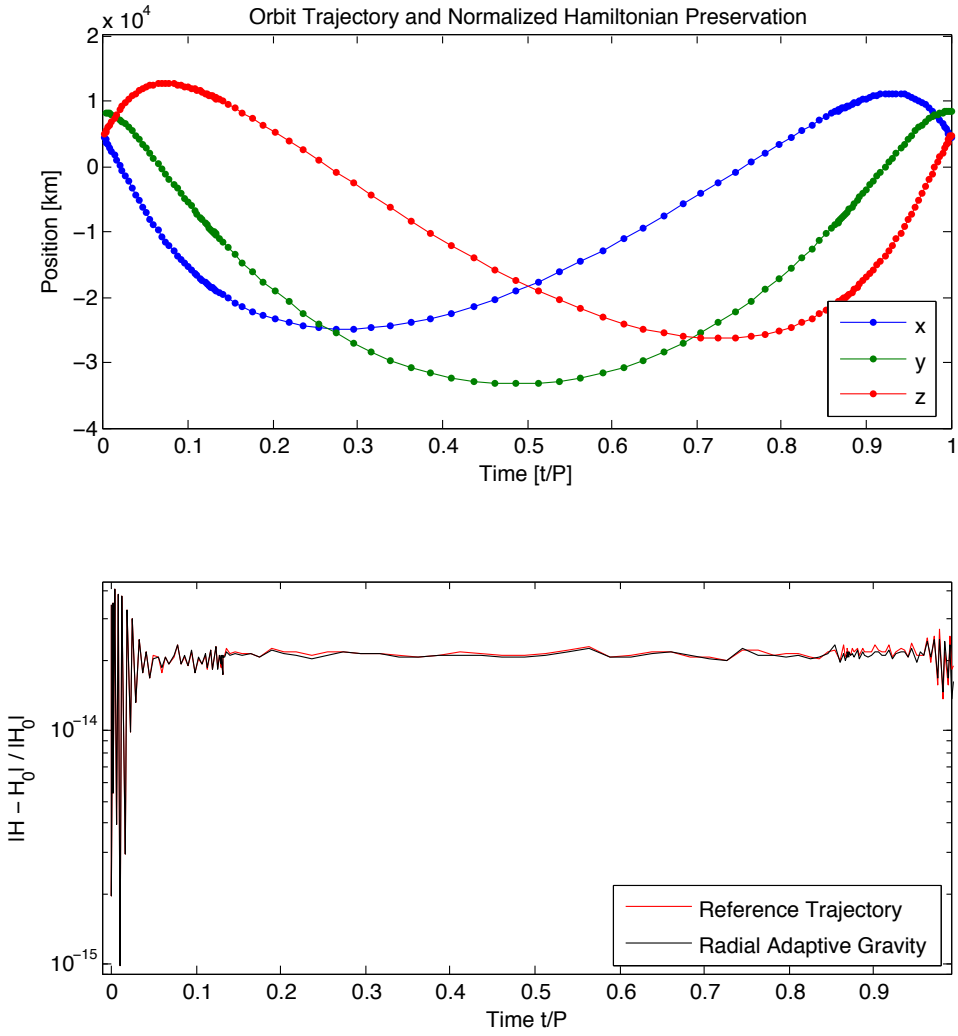


Figure 2.8: Radially adaptive gravity method - orbit position states and normalized Hamiltonian preservation along GEO transfer orbit.

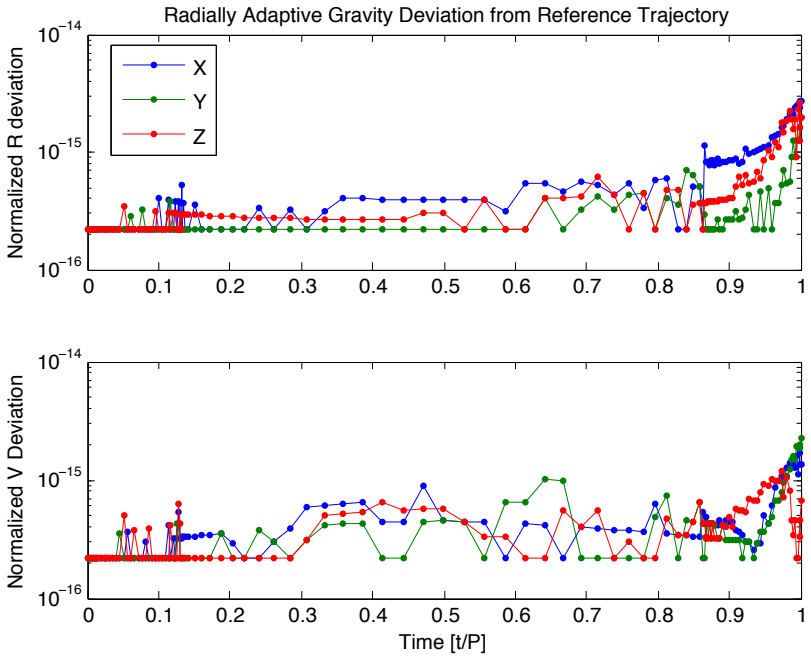


Figure 2.9: Radially adaptive gravity method - deviation from reference trajectory along GEO transfer orbit.

error), it is computationally wasteful to calculate a full accuracy spherical harmonic gravity series at each evaluation point. Towards the end of the iteration process, when MCPI has converged in position/velocity accuracy to seven or more digits, the trajectory itself is not spatially changing very much from iteration to iteration, but higher accuracy in acceleration is required to converge further. The essence of the multiple fidelity force model strategy is to calculate local perturbation forces (in the vicinity of each node) only as accurately as is required to keep MCPI smoothly converging, or about one or two digits more precisely than the local solution accuracy. This approach is generally valid for all environmental perturbation forces, but we focus on gravitational forces here as they are the most computationally expensive in most cases.

The multiple fidelity gravity model method is shown in Figure 2.10. Three gravity models of varying complexity are utilized throughout the process of Picard iteration, the simplest being the first five zonal harmonics J_2 through J_6, which are computationally inexpensive to calculate in comparison to full order spherical harmonic gravity. During the Picard iteration in which a higher fidelity force model is used for the first time, a local Taylor series correction at each node is calculated that locally approximates the higher fidelity model to sufficient precision without requiring further "full" force evaluations. Subsequent Picard iterations do not require the calculation of the higher fidelity gravity model, rather an evaluation of the J_2 through J_6 model, to which the Taylor correction is added to account locally for the difference between the approximate and the high-order gravity model. An *a priori* study can rigorously establish the validity of these approximations as a function of the local displacement from the nodal coordinates where the Taylor series approximation is calculated with the full order model.

The full-order gravitational acceleration $\boldsymbol{g_F}(\boldsymbol{r})$ at a point $\boldsymbol{r} = [x, y, z]^T$ is a spherical harmonic series with some conservatively specified maximum degree and order. Typically the maximum degree/order value is chosen based upon prior insights into the desired fidelity of the output solution, the computational resources available, and the orbital regime of interest. The methods underlying the radially adaptive gravity method of Section 2.2 can be invoked to make these traditionally heuristic decisions more rigorous. For the present discussion, we adopt a low order approximate gravity model: the gravitational acceleration due to two-body Keplerian motion plus the contributions of the first five zonal harmonic gravity terms J_2 through J_6, which we designate $\boldsymbol{g_z}(\boldsymbol{r})$. The difference between whatever full-order gravitational acceleration model is selected and the zonal approximate gravitational acceleration is Taylor

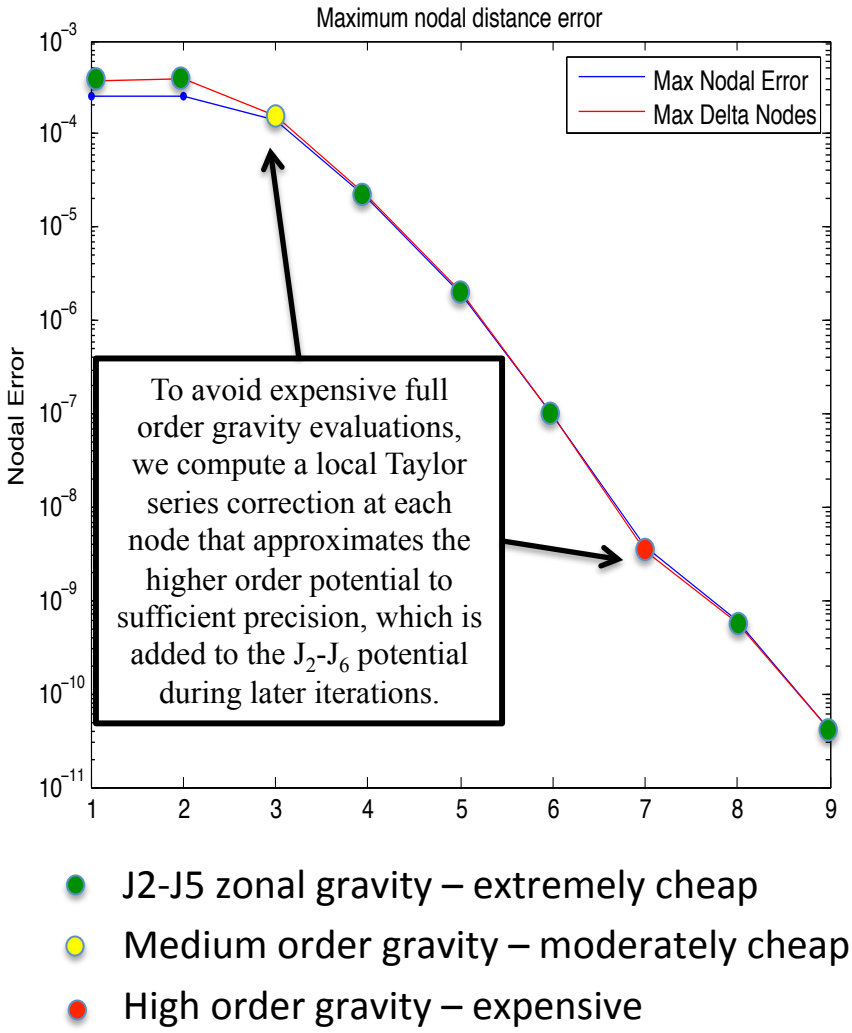


Figure 2.10: Representative MCPI convergence error history using the multiple-fidelity gravity method.

expanded at an arbitrary point $\boldsymbol{r_1}$ near the initial point $\boldsymbol{r_0}$ as

$$\begin{aligned}[\boldsymbol{g_F}(\boldsymbol{r_1}) - \boldsymbol{g_z}(\boldsymbol{r_1})] \simeq [\boldsymbol{g_F}(\boldsymbol{r_0}) - \boldsymbol{g_z}(\boldsymbol{r_0})] + \\ \boldsymbol{\nabla}\left[\boldsymbol{g_F}(\boldsymbol{r_0}) - \boldsymbol{g_z}(\boldsymbol{r_0})\right][\boldsymbol{r_1} - \boldsymbol{r_0}] + H.O.\end{aligned} \tag{2.9}$$

Re-arranging the above equation and neglecting the higher-order terms provides an approximation of the full-order gravity $\boldsymbol{g_F}(\boldsymbol{r_1})$ as

$$\boldsymbol{g_F}(\boldsymbol{r_1}) = \boldsymbol{g_z}(\boldsymbol{r_1}) + \boldsymbol{c_0} + [A_0][\Delta \boldsymbol{r}] + Truncation\ Error \tag{2.10}$$

where $\boldsymbol{c_0} = [\boldsymbol{g_F}(\boldsymbol{r_0}) - \boldsymbol{g_z}(\boldsymbol{r_0})]$ is a constant vector quantity evaluated at the Taylor series expansion point, $A_0 = \boldsymbol{\nabla}\left[\boldsymbol{g_F}(\boldsymbol{r_0}) - \boldsymbol{g_z}(\boldsymbol{r_0})\right]$ is the gradient matrix at the expansion point, and $\Delta \boldsymbol{r} = [\boldsymbol{r_1} - \boldsymbol{r_0}]$ is the position difference vector. For a given Picard iteration process, Equation 2.10 will be applied at each node along the approximate trajectory to enable subsequent Picard iterations. A zeroth order Taylor approximation may be applied by considering only the terms $\boldsymbol{g_F}(\boldsymbol{r_1}) = \boldsymbol{g_z}(\boldsymbol{r_1}) + \boldsymbol{c_0}$ in the approximation of full order gravity, whereas a first order Taylor approximation method utilizes all the terms in Equation 2.10. In all cases, *a priori* computations can bound worse case truncation errors as a function of $|\boldsymbol{r_0}|$ and $|\boldsymbol{\Delta r}|$.

Remarkably, MCPI is able to achieve a large computational speedup by using a zeroth order Taylor series approximation in the multiple fidelity gravity method, while still being able to achieve arbitrarily high accuracy. This is because $\boldsymbol{\Delta r}$ is small, and over small regions the discrepancy between $\boldsymbol{g_z}(\boldsymbol{r})$ and $\boldsymbol{g_F}(\boldsymbol{r})$ is approximately constant at each node. Figure 2.11 shows a comparison of computation time and function evaluation count for MCPI and other state of practice numerical propagators, applied to an eccentric orbit with a perigee radius in LEO [50]. In these plots, MCPI without the multiple fidelity gravity method is labeled "MCPI",

and MCPI with the zeroth order Taylor series multiple fidelity method is labeled "TCA-MCPI". The zeroth order Taylor approach is also generally useful during the analogous convergence process of Implicit Runge Kutta propagators [15–17].

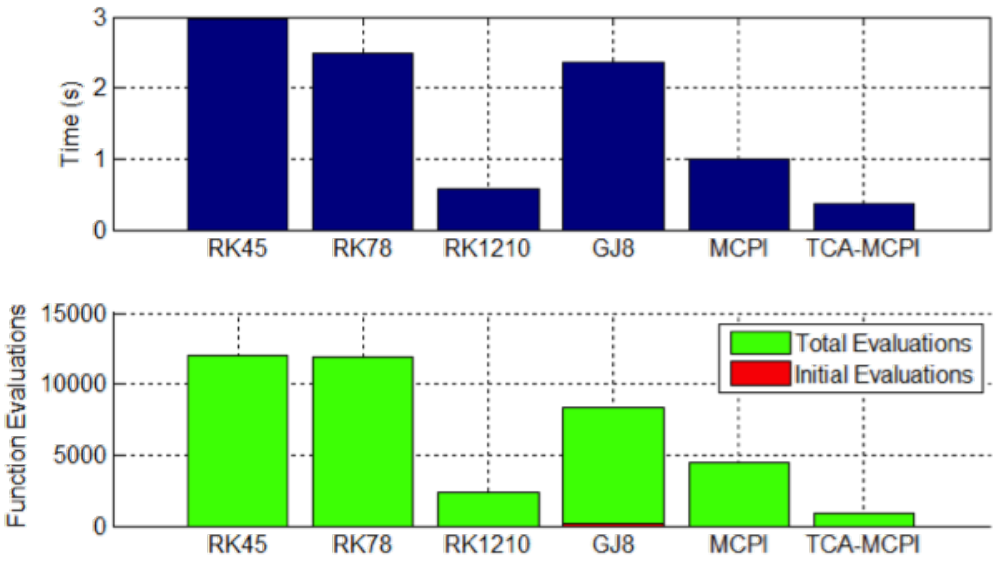


Figure 2.11: MCPI zeroth order Taylor series propagation, computational performance compared to state of practice numerical propagators for eccentric LEO orbit.

Within a single instance of MCPI, utilizing a first order Taylor approximation does not provide much improvement over the zeroth order approximation because the nodal corrections between iterations are generally small, especially at the end of the convergence process when the highest approximation accuracy is required. However, an important benefit of the first order Taylor method is that it can increase the domain of validity of the Taylor approximation beyond that of the zeroth order method. This means that after having calculated the local Taylor approximation to the gravity field once, neighboring trajectories within the domain of validity may be

propagated essentially "for free". This is ideal for propagating particle trajectories in the vicinity of a reference trajectory, for instance within Monte Carlo analysis or uncertainty propagation, or within a particle or Sigma-Point filter. There is, however, a moderate computational penalty when propagating the reference trajectory, as the gradient A_0 matrix must be calculated.

The gradient matrix A_0 can be computed analytically during the propagation of the reference trajectory (within the MCPI convergence loop) by taking partial derivatives of the spherical harmonic series, which adds an additional computational penalty beyond the cost of the series itself (estimated 15-20% extra computation time). These analytic partial derivatives are the same computations that are required to numerically propagate the perturbed state transition matrix[2], and are included in a forthcoming version of MCPI [51]. In this case, the position difference vector $\boldsymbol{\Delta r}$ comes from the nodal corrections between Picard iterations.

Alternatively, the gradient matrix A_0 may be computed numerically by performing manual perturbations away from a reference trajectory and solving the system of equations. This method was first presented earlier this year [52].

To numerically solve for the gradient matrix, MCPI is first used to propagate a high-fidelity reference trajectory $\boldsymbol{r}(t)$. A Taylor expansion about time-sampled positions $\boldsymbol{r_0}$ along the reference trajectory is given by Equation 2.10, which can be rearranged to give

$$[\boldsymbol{g_F}(\boldsymbol{r_1}) - \boldsymbol{g_z}(\boldsymbol{r_1})] - \boldsymbol{c_0} = [A_0][\Delta \boldsymbol{r}] \tag{2.11}$$

Evaluating the full-order gravity function $\boldsymbol{g_F}(\boldsymbol{r})$ and the zonal gravity function $\boldsymbol{g_z}(\boldsymbol{r})$ at the point $\boldsymbol{r_1}$ means the left hand side of Equation 2.11 is a known quantity, which

[2]Credit is due to my colleague Julie Read, who has painstakingly derived, tested, and documented the analytic partial derivatives of the spherical harmonic series for propagation of the state transition matrix with MCPI.

is renamed $[\Delta \boldsymbol{g_1}]$. This gives

$$[\Delta \boldsymbol{g_1}] = [A_0][\Delta \boldsymbol{r}] \tag{2.12}$$

where $[\Delta \boldsymbol{g_1}]$ and $[\Delta \boldsymbol{r}]$ are known (3x1) vectors, and the gradient matrix $[A_0]$ is an unknown 3x3 symmetric matrix. The matrix system in Equation 2.12 can be transmuted in terms of the six unknown elements of the $[A_0]$ gradient matrix:

$$\begin{bmatrix} \Delta g_{x1} \\ \Delta g_{y1} \\ \Delta g_{z1} \end{bmatrix} = \begin{bmatrix} \Delta x_1 & \Delta y_1 & \Delta z_1 & 0 & 0 & 0 \\ 0 & \Delta x_1 & 0 & \Delta y_1 & \Delta z_1 & 0 \\ 0 & 0 & \Delta x_1 & 0 & \Delta y_1 & \Delta z_1 \end{bmatrix} \begin{bmatrix} A_{11} \\ A_{12} \\ A_{13} \\ A_{22} \\ A_{23} \\ A_{33} \end{bmatrix} \tag{2.13}$$

Equation 2.13 is a matrix system of the form $\Delta \boldsymbol{g} = B\boldsymbol{a}$, where the terms Δx_1, Δy_1, and Δz_1 forming the B matrix are the elements of the $[\Delta \boldsymbol{r}]$ vector. Three independent offsets from the reference trajectory to positions $\boldsymbol{r_1}$, $\boldsymbol{r_2}$, and $\boldsymbol{r_3}$ are performed. Stacking the transmuted system of Equation 2.13 from the three offsets gives $\Delta \boldsymbol{g} = [\Delta \boldsymbol{g_1}; \Delta \boldsymbol{g_2}; \Delta \boldsymbol{g_3}]^T$ which is a (9x1) vector, B which is a (9x6) matrix, and a which is a (6x1) vector containing the unknown elements of the A_0 matrix. This system can be solved using the normal equations of least squares

$$\boldsymbol{a} = (B^T B)^{-1} B^T \Delta \boldsymbol{g} \tag{2.14}$$

to find the best estimate of the unknown elements a_i of the A_0 matrix [53]. It would seem that two offsets away from the reference trajectory would provide a (6x6) B

matrix that could be inverted directly to solve for the $\boldsymbol{a}$ vector, but in practice this (6x6) B matrix is singular and non-invertible. This is easily cured by introducing one or two additional evaluations and using a least square inverse.

This first order Taylor series gravity approximation is ideal for propagating trajectories in the vicinity of the reference trajectory, such as in a Monte Carlo analysis or during the process of uncertainty propagation. The gradient matrix along the reference trajectory can be calculated ahead of time allowing neighboring trajectories to be propagated without performing any high-fidelity gravity evaluations during the MCPI iteration process. This means that using MCPI to propagate trajectories near the reference trajectory can be done to within the accuracy limit of the local first order Taylor approximations nearly instantaneously. Figure 2.12 shows a trajectory propagated using the first order Taylor approximation near a GEO reference trajectory of the Telemetry Data Relay Satellite (TDRS-11) over a period of 20 days (20 orbits). Figure 2.12a is the spatial deviation of the orbit away from the reference orbit, which varies between roughly two to ten kilometers, starting from an initial deviation of roughly 3.5 kilometers. Figure 2.12b shows the Hamiltonian preservation of the numerically propagated reference trajectory, and of the neighboring trajectory propagated using the first order Taylor series gravity approximation. The Taylor series solution is able to conserve the Hamiltonian to a precision of about 10^{-12} for this near-GEO case.

The same plots for a LEO orbit similar to that of the International Space Station (ISS), with an orbital altitude of roughly 420km, are shown in Figure 2.13. Using the same initial deviation of roughly four kilometers, the first order Taylor series gravity is able to preserve the Hamiltonian to a precision of about 10^{-9} over 20 orbits (1.25 days). Note that for the Low-Earth Orbit ISS-like case, an initial deviation of 3.5 kilometers is a much larger relative deviation when normalized by the orbital radius

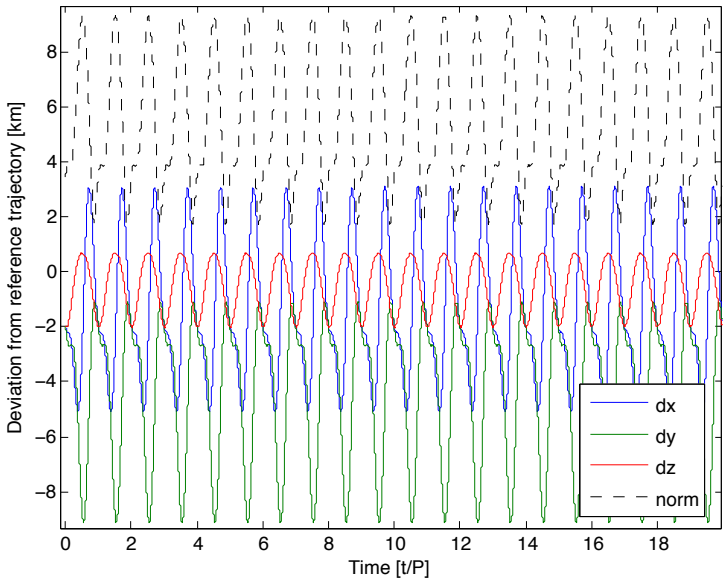
than it is for the GEO orbit which has a semi-major axis of about 42 thousand kilometers. Furthermore the LEO case gravity model is much more nonlinear than the GEO case.

The Hamiltonian preservation is not a perfect measure of the obtainable solution accuracy of a numerical propagator (especially for these near circular reference orbits), but it is a reasonable metric to show that high fidelity solutions can be attained with vastly reduced computational cost. If more precision is required from the final propagated solution after the first order Taylor series gravity has converged to its achievable accuracy limit, further MCPI iterations using the zeroth order Taylor series may be performed to achieve arbitrarily accurate trajectories.

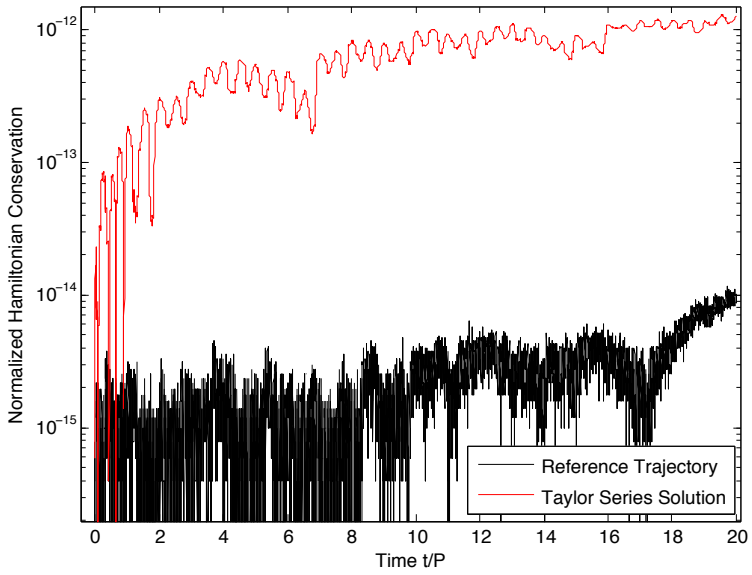
2.4 Cold/Warm/Hot Start

MCPI, when applied to the problem of perturbed orbit propagation, has been shown to be able to converge for segments lengths of up to several orbital periods, even with a completely uninformed initial approximation $\boldsymbol{x}^0(\tau)$ [32]. Typically, if no *a priori* approximation of the system state trajectories is available, the value of the states at all CGL nodes is initially set equal to the boundary conditions. This method is called the "cold-start" method. However, any initial approximation to the true dynamics will allow MCPI to converge to the true solution in fewer iterations. If the initial approximation is sufficiently good (with, say, 10^{-4} or smaller relative errors) then we can immediately invoke the force approximation ideas from the previous section, and typically need only one full force evaluation per node to achieve final convergence.

The equations of generally perturbed orbital motion are given by Equation 2.1. The dominant acceleration force acting on an object in Earth orbit at instantaneous position $\boldsymbol{r}$ is the Keplerian gravity term $\ddot{\boldsymbol{r}} = -\frac{\mu}{r^3}\boldsymbol{r}$. Added to this are other dis-

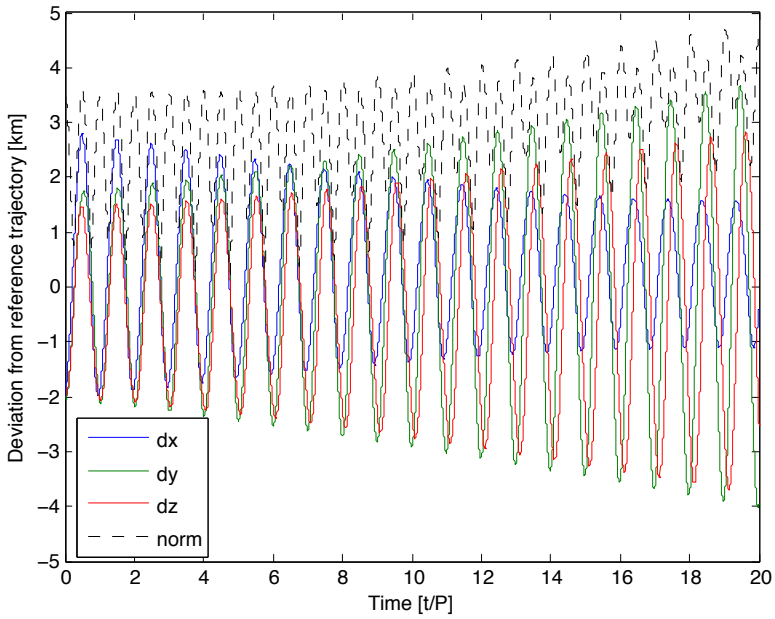


(a) Spatial deviation of first order Taylor series trajectory in neighborhood of TDRS-11 reference orbit.

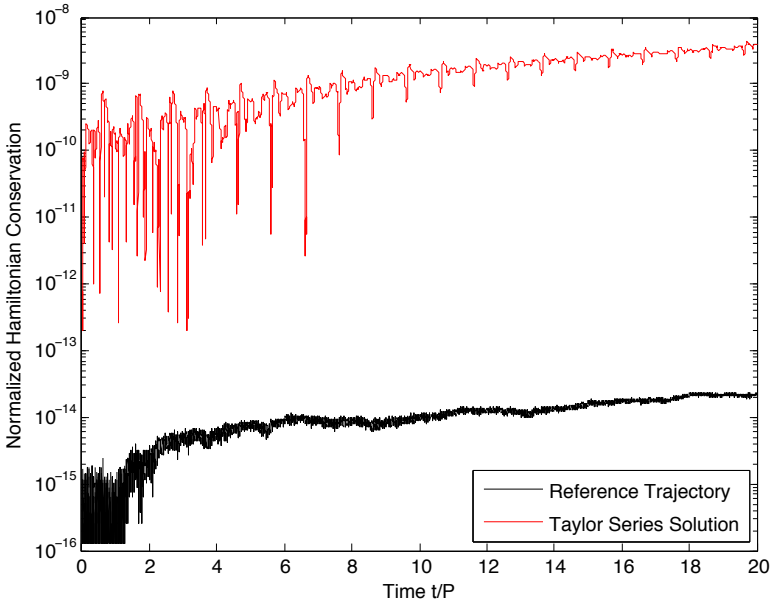


(b) Hamiltonian preservation of reference orbit and neighboring trajectory with first order Taylor gravity approximation.

Figure 2.12: First order Taylor series gravity approximation for 20 day numerical propagation near GEO reference orbit.



(a) Spatial deviation of first order Taylor series trajectory in neighborhood of ISS reference orbit.



(b) Hamiltonian preservation of reference orbit and neighboring trajectory with first order Taylor gravity approximation.

Figure 2.13: First order Taylor series gravity approximation for 20 orbit numerical propagation near ISS reference orbit.

turbance forces $\boldsymbol{a_d}$ caused by the gravitation potential of the non-spherical Earth, atmospheric drag, third body gravitational effects of the Sun and Moon, solar radiation pressure, and other complex forces due to the physics of motion in the space environment. Approximate solutions to the generally perturbed Earth-orbiting satellite problem are available. It is logical to invoke these and other insights to establish a better starting orbit approximation than the cold-start method. Implicit Runge Kutta (IRK) algorithms have previously utilized analytic and semi-analytic methods to provide an initial estimate for perturbed orbit propagation [16, 19].

A two-stage initial orbit approximation method can greatly reduce the amount of computation MCPI requires for perturbed orbit propagation of more than one orbit period. The first step is to "warm-start" MCPI at the CGL nodes during the first lap around the orbit using the analytic solution for Keplerian motion or a semi-analytic solution for perturbed motion. This won't be perfect because the perturbation forces will cause the true motion to deviate from the approximate solution, however typically a starting estimate accurate to three or more significant digits will result. Allowing MCPI to converge and then saving the difference between the warm start estimate and the final converged solution gives an Encke-type description of the deviation (at the nodes) of the true motion from the starting approximation. The next lap around the orbit, MCPI can be warm started with the same approximate solution, and then the deviation from the previous warm start can be applied on top of that to give the "hot-start".

The motivation for this idea is that the non two-body perturbations are highly correlated on successive orbits. Asking the question "How did the previous warm start differ from the final convergence?" is useful to trend-sense a correction for the next pass through a neighboring part of the force field. In LEO, the short orbital period means that perturbations are similar on each trip around the orbit relative

to the longer timescale of the Earth rotation. Near GEO, the dominant perturbation terms are the first few zonal harmonic gravity terms, which are symmetric with respect to Earth rotation. Typically the hot-start process gives a starting approximation with errors in the 5th significant figure or better. This process is illustrated in Figure 2.14. Note that this figure is not drawn to scale, the deviations are many orders of magnitude smaller than the orbit radius in practice.

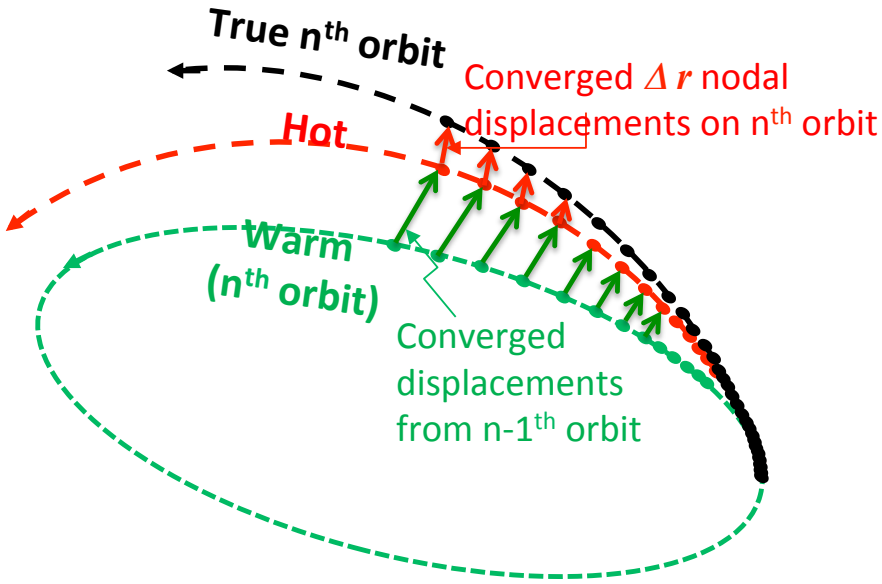


Figure 2.14: Heuristic schematic of cold/warm/hot-start methods for MCPI perturbed orbit propagation.

A reasonable starting approximation to the true dynamics can be obtained by the analytic solution to the problem of Keplerian motion, the Lagrange/Gibbs F and G solution. In the absence of disturbance forces $\boldsymbol{a_d}$, the orbital motion will forever be contained within a fixed plane, a vector space which is spanned by two non-parallel vectors. Any point on the orbit may be located as a linear combination of the two

vectors spanning the space. The F and G series uses the position and velocity vectors $\boldsymbol{r}(t_0)$ and $\dot{\boldsymbol{r}}(t_0)$ at some arbitrarily specified initial time t_0 as the vectors with which to span the space. Thus the position and velocity at any time are described by a linear combination of the basis vectors

$$\boldsymbol{r}(t) = F\boldsymbol{r}(t_0) + G\dot{\boldsymbol{r}}(t_0) \tag{2.15a}$$

$$\dot{\boldsymbol{r}}(t) = \dot{F}\boldsymbol{r}(t_0) + \dot{G}\dot{\boldsymbol{r}}(t_0) \tag{2.15b}$$

where $(F, G, \dot{F}, \dot{G})$ may be solved in terms of the classical orbital elements and the given initial conditions [41].

A better approximation to the dynamics of perturbed orbit approximation may be obtained with a semi-analytic solution, which takes into account the time-averaged secular effects of a subset of the disturbing accelerations. Presented by Kozai [54], and independently (in the same issue of the same journal) by Brouwer [55], the original semi-analytic models include the time averaged effect of the zonal harmonic gravity terms up to J_5. Brouwer theory makes use of Delaunay orbital elements, and was originally vulnerable to singularity effects for perfectly circular orbits, zero inclination orbits, and critically inclined orbits. Subsequent work added drag and third-body perturbation effects to the analysis, and addressed the issue of singularities in Brouwer theory [56, 57]. The U.S. Air Force developed the *Simplified General Perturbations 4* (SGP4) method based upon Brouwer theory, for which the equations and source code were released in 1980 [58]. The U.S Navy independently developed a method based upon Brouwer-Lyddane theory called *Position Partials and Time* (PPT3) [59]. More recently, Vallado has revisited and clarified SGP4, providing open source code, test cases, and documentation [60].

The analytic solution of Keplerian motion, or any of the semi-analytic methods,

may be used to warm- and hot-start MCPI by providing an initial trajectory estimate. Figure 2.15 demonstrates the effectiveness of the warm- and hot-start methods using the F&G analytic solution. In this figure, the orbit being propagated is a circular GEO orbit, with spherical harmonic gravity as the only perturbation. With no *a priori* information ("cold-start"), MCPI is able to converge in 14 iterations, from a large normalized relative nodal error (normalized by the initial conditions) on the order of 1 (since the initial boundary condition values is used at all nodes). This is indicated by the black curve in the figure. Warm-starting each node along the trajectory with the analytic F and G solution brings the initial error down three orders of magnitude, to around 10^{-3}. MCPI is able to converge in 11 iterations, as indicated by the blue curve. After propagating all the way around the orbit a single time, the initial trajectory estimates on the subsequent orbit can be hot-started. This provides an initial error on the order of 10^{-5} and allows MCPI to converge in 9 iterations, as indicated by the red curve. Eliminating iterations means eliminating expensive force function evaluations and computational overhead. It also means that the local force approximations (from Section 2.3) can be invoked to make all subsequent iterations extremely inexpensive. Warm-starting on the first trip around the orbit is able to reduce computational cost by roughly 20%, and hot-starting on subsequent trips around the orbit reduces cost by roughly 35%. Invoking local force approximations greatly reduces the cost further.

If the error tolerance for the results shown in Figure 2.15 were set to 10^{-8} (sub-meter precision) instead of 10^{-12}, hot-started MCPI converges in 5 or 6 iterations. As discussed in Section 2.3, this can readily be accomplished with *only one expensive "full" gravity evaluation* per node due to the fixed point nature of MCPI convergence. Utilization of local force approximations dramatically accelerates all of the computations after a relative error of 10^{-5} is achieved, and the hot-start process usually

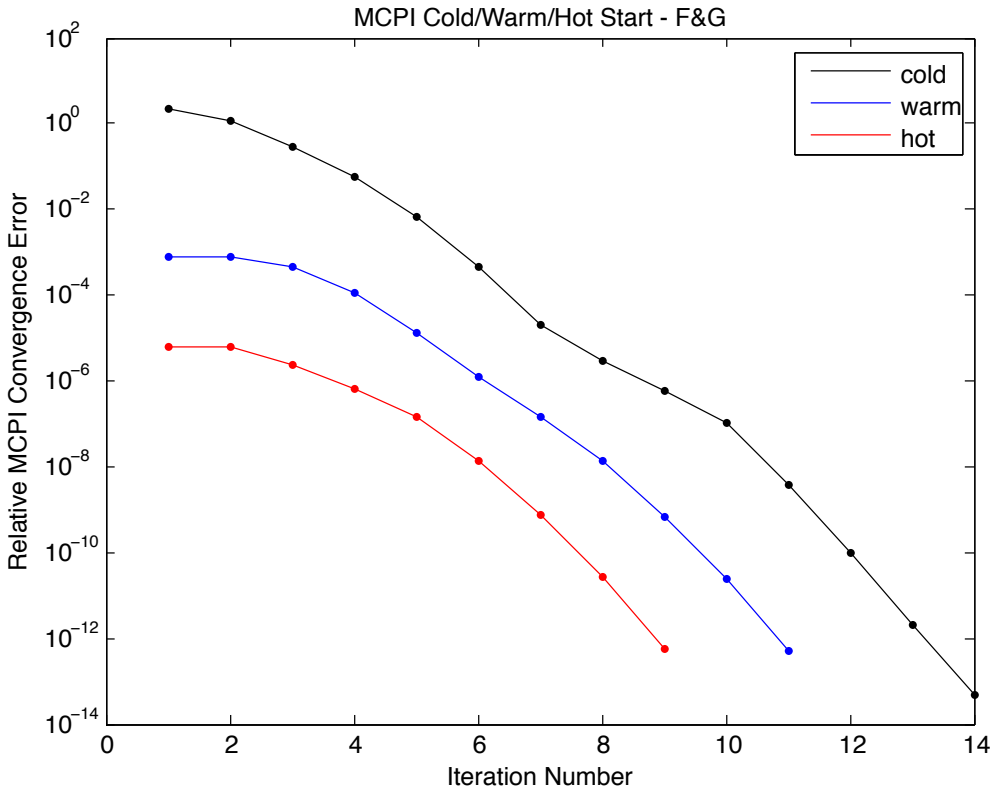


Figure 2.15: MCPI cold/warm/hot-start convergence trends using F&G analytic solution on GEO orbit.

allows this on the very first iteration. Thus in Figure 2.15 we see that the hot start process converges to engineering precision (sub-meter orbit: $\sim 10^{-8}$ relative error) in only 6 iterations, and only the first of these requires a full force evaluation at each node. This is the key to computational efficiency.

The F&G analytic (Keplerian) solution does not take into account any perturbations. The semi-analytic solutions, for instance SGP4, take into account a time-averaged effect of zonal harmonic gravity, simple drag, and third-body sun and moon effects. Depending upon the perturbation models considered in the numerical integration, and the orbital regime of interest, the semi-analytic methods can provide better warm/hot start initial approximations than the analytic method. However, the analytic method is much simpler to work with. The semi-analytic methods require careful transformation between osculating states (in which the numerical integrator works) and the time-averaged states (in which the semi-analytic methods work). Additionally, different semi-analytic methods work with different time-averaged elements. For reasons of simplicity, the F&G analytic warm/hot start is preferable for stand-alone numerical integration tools. However, if working with (for instance) a legacy toolset that already includes semi-analytic (general perturbations) code, the existing semi-analytic methods may be used to warm/hot start MCPI with better initial estimates than the F&G two-body warm start would provide.

2.5 First Order Cascade MCPI

The equations of perturbed orbital motion (Equation 2.1) are a non-linear, second order, vector differential equation. An elegant second order MCPI framework was presented in Section 1.4.4 (based upon Bai's original second order method [31, 32]) to directly integrate second order systems. If, however, for some reason it is desired to use the traditional first order MCPI on a second order system, the system may be

decomposed into a set of first order differential equations. Doing so for the equations of perturbed orbital motion gives

$$\boldsymbol{z}(t) = \begin{bmatrix} \boldsymbol{r}(t) \\ \boldsymbol{v}(t) \end{bmatrix}, \quad \boldsymbol{z_0} = \begin{bmatrix} \boldsymbol{r_0} \\ \boldsymbol{v_0} \end{bmatrix} \tag{2.16a}$$

$$\dot{\boldsymbol{z}} \equiv \boldsymbol{F}(t, \boldsymbol{z}(t)) = \begin{bmatrix} \boldsymbol{v} \\ -\frac{\mu}{r^3}\boldsymbol{r} + \boldsymbol{a_d} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{f}(t, \boldsymbol{z}(t)) \end{bmatrix} \tag{2.16b}$$

where the new augmented state vector $\boldsymbol{z}$ is of length $n = 6$. As discussed in Section 1.4.4, this method is computationally more expensive than the second order MCPI framework because the position and velocity terms are independently approximated with Cheybshev polynomials thus doubling the dimensionality of the problem. However, if is undesirable to use the second order formulation, for instance if the developer is implementing MCPI in an existing code base with a first order architecture (i.e. the equations naturally occur in the $\dot{\boldsymbol{z}} = \boldsymbol{f}(t, \boldsymbol{z}(t))$ form), the first order approach can be used. However, care must be taken.

To demonstrate this, consider the system in Equation 2.16b. Picard iteration updates for the augmented state $\boldsymbol{z}$ are given by

$$\boldsymbol{z}^i(\tau) = \boldsymbol{z_0} + \int_{-1}^{\tau} \boldsymbol{g}(s, \boldsymbol{z}^{i-1}(s))ds \quad i = 1, 2, ... \tag{2.17}$$

where the integrand $\boldsymbol{g}(\tau, \boldsymbol{z}^{i-1}(\tau))$ is a result of approximating each term in $\boldsymbol{F}(t, \boldsymbol{z}(t))$ on the right hand side of Equation 2.16b with an $(N-1)^{th}$ order Chebyshev polynomial series. Explicitly considering all the terms in the Picard update of the aug-

mented system gives

$$\begin{bmatrix} \boldsymbol{r}^i(\tau) \\ \boldsymbol{v}^i(\tau) \end{bmatrix} = \begin{bmatrix} \boldsymbol{r_0} \\ \boldsymbol{v_0} \end{bmatrix} + \int_{-1}^{\tau} \begin{bmatrix} \boldsymbol{v}^{i-1}(s) \\ \boldsymbol{g}(s, \boldsymbol{r}^{i-1}(s), \boldsymbol{v}^{i-1}(s)) \end{bmatrix} ds \tag{2.18}$$

Updates to position states $\boldsymbol{r}^i$ come from the $(i-1)^{th}$ velocity estimates $\boldsymbol{v}^{i-1}$, which is a purely kinematic relationship. Velocity updates come from the $(i-1)^{th}$ position and velocity estimates $\boldsymbol{r}^{i-1}$ and $\boldsymbol{v}^{i-1}$, which is dictated by the physics of the problem. Therefore, forcing the second order system into first order Picard iteration form has introduced a *one-iteration phase lag* between the state updates, such that corrections to either $\boldsymbol{r}$ or $\boldsymbol{v}$ are not reflected in the other set $\boldsymbol{v}$ or $\boldsymbol{r}$ until the next iteration. We end up with a "stutter-step" convergence pattern of updating one set of states at a time per iteration, yet calculating the full force function evaluations $\boldsymbol{g}(\tau, \boldsymbol{z}^{i-1}(\tau))$ every iteration, which is computationally very wasteful (experience indicates a doubling of the number of iterations).

This trend is very apparent in Figure 2.16, which shows the convergence history for a single MCPI segment of an ISS-like Low Earth Orbit, starting from an F and G warm start. The jagged red line is the normalized MCPI convergence error plotted with respect to iteration count. MCPI takes 23 iterations to converge to within the convergence threshold, shown by the horizontal green line in the plot. Although the red curve looks chaotic, it is actually the super-position of two independent, smooth curves; one for the position states convergence, and one for the velocity states convergence. These curves are plotted as the dashed and dot-dashed black lines. It is apparent that each set of states $\boldsymbol{r}$ and $\boldsymbol{v}$ are updating every other iteration. The remedy for this is to combine two "half iterations" into each Picard iteration, a method we call first order Cascade MCPI.

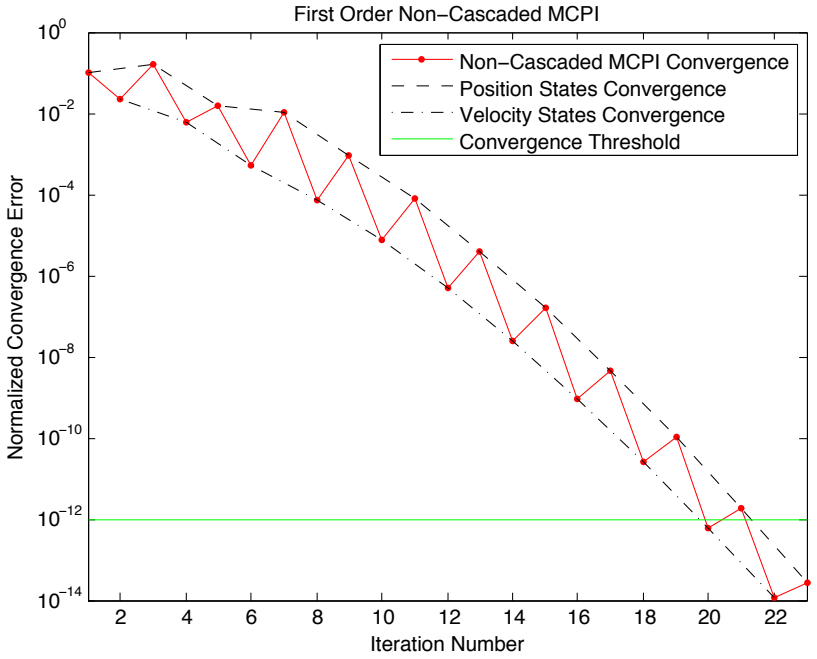


Figure 2.16: Non-cascade first order MCPI convergence for single segment of LEO (ISS-like) orbit.

The first half iteration is to update the velocity states using the lower half of Equation 2.18:

$$\boldsymbol{v}^i(\tau) = \boldsymbol{v_0} + \int_{-1}^{\tau} \boldsymbol{g}(s, \boldsymbol{r}^{i-1}(s), \boldsymbol{v}^{i-1}(s))ds \tag{2.19}$$

Having updated the velocity states, the second half iteration is to update the position states from the updated velocity states using the kinematic constraint in the top half of Equation 2.18:

$$\boldsymbol{r}^i(\tau) = \boldsymbol{r_0} + \int_{-1}^{\tau} \boldsymbol{v}^i(s)ds \tag{2.20}$$

In the vector/matrix notation of Section 1.4.3, these operations are

$$Z_v^i = C_x C_\alpha G(Z^{i-1}) + C_x X_{0,v} \tag{2.21a}$$

$$Z_r^i = C_x C_\alpha Z_v^i + C_x X_{0,r} \tag{2.21b}$$

where we have implicitly defined an augmented state matrix $Z^i \equiv [X_r^i, X_v^i]$ by combining the position and velocity state matrices. $X_{0,v}$ and $X_{0,r}$ are the boundary condition constraint matrices for the velocity and position states, respectively, and $G(Z^{i-1})$ is the stacked matrix form of the full acceleration functions $\boldsymbol{g}(\tau, \boldsymbol{z}^{i-1}(\tau))$.

Using the Cascade first order MCPI method for the same orbit segment as was shown in Figure 2.16 gives the convergence trend in Figure 2.17. There is only one smooth curve, which achieves convergence in 11 iterations, as opposed to the 23 iterations required with the non-Cascade method. This means that the Cascade method has effectively cut the required number of expensive full force evaluations $\boldsymbol{g}(\tau, \boldsymbol{z}^{i-1}(\tau))$ in half.

Note that this is different than the second order MCPI formulation in Section 1.4.4 because the position and velocity states are independently approximated. The explicit kinematic relationship between the coefficients of the velocity states and the coefficients of the position states (that characterized the second order method in Section 1) is not present here. The kinematic relationship between the states is here enforced by the augmented differential equation in Equation 2.16b. This is not the preferred method to use for a naturally second order system, since the dimensionality of the problem is doubled by independently approximating the position and velocity states. If however, for implementation reasons, the system must be decomposed into a set of first order equations, this Cascade method is essential to achieve efficiency. Not using the Cascade version will result in a stutter-step convergence pattern be-

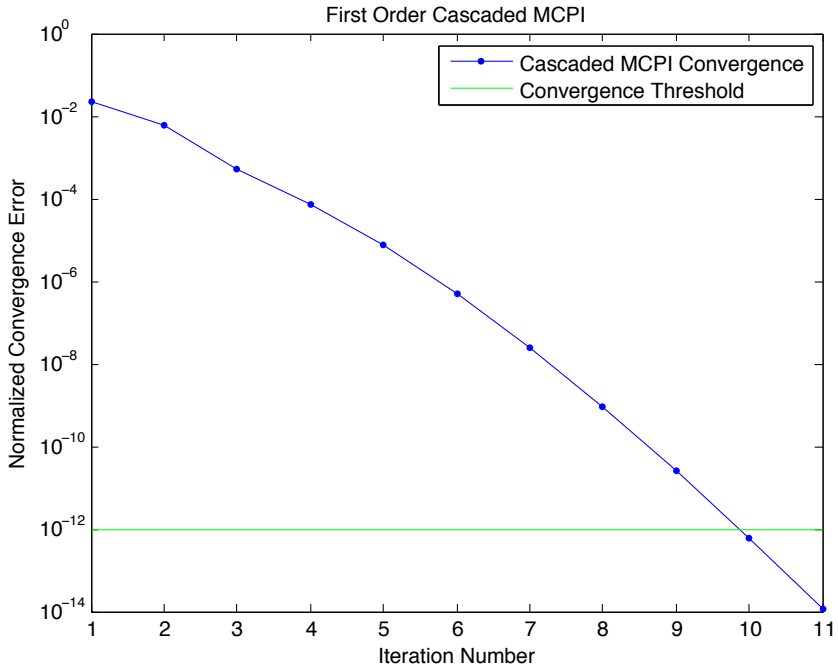


Figure 2.17: Cascade first order MCPI convergence for single segment of LEO (ISS-like) orbit.

tween the position and velocity states, requiring roughly twice the number of Picard iterations as shown in Figures 2.16 and 2.17.

3. MCPI TUNING PARAMETER AUTOMATION

3.1 Introduction

Until recently MCPI tuning parameters had to be chosen by the user by hand, and generally take different values for each application of MCPI. Even for a single application, such as perturbed orbit propagation, the optimal parameters will vary based upon the shape of the orbit in question and the perturbations being considered. The primary tuning parameters are *(i)* the accuracy tolerance (desired number of significant figures of the solution), *(ii)* the force model parameters (e.g. the maximum degree and order of the spherical harmonic gravity model), *(iii)* the maximum degree N of the Chebyshev polynomials used in the path approximation, and *(iv)* the break times of the sequence of path segments. The first two are frequently set as constraints while the latter two are optimized.

This section presents recent work that serves to automate the tuning parameter selection process so that MCPI may be used to propagate perturbed orbits with a given accuracy requirement, without a user in the loop [61]. Two *a priori* tuning parameter value sets are presented; the first is an empirically generated "safe" set that is guaranteed to meet the accuracy requirement but may deliver suboptimal algorithmic run-time performance, and the second is a more optimal set generated procedurally using a genetic algorithm and which delivers slightly higher algorithmic performance and meets the accuracy requirement for an ensemble average of orbits. These initial parameter set selection logic schemes are designed to choose a reasonable set of tuning parameters based upon the properties of the orbit being propagated. However, the parameters can (and should) be adapted locally during the convergence process, as needed, in order to maintain the accuracy tolerance in

the presence of other force functions. An adaptive algorithm is described to locally vary the parameters from these *a priori* chosen parameter sets. Whereas the two initial tuning parameter sets are particular to perturbed orbital motion, this adaptive algorithm is generally applicable to any ODE system.

3.2 MCPI Parameters

3.2.1 Description of Parameters

Number of Segments Per Orbit To propagate an ephemeris forward or backward, the ephemeris period is broken up into segments $s_1, s_2, ..., s_n$ each with corresponding time intervals consistent with the finite interval $(t_f - t_0) < \delta$ of MCPI convergence. In previous orbit propagation studies MCPI has been shown to be capable of convergence over long segment lengths, even several orbit periods long if desired [32]. For this study, we shall restrict the segment length to be less than or equal to one orbital period, so that we have at least one, but generally more than one segment per orbit. Note that this statement is not restricting the total ephemeris propagation time, which may be any arbitrary period, but rather making the restriction that there shall be $n \geq 1$ segments per orbit period. The segments of the ephemeris fit together "daisy chain" style with segment dependent tuning to ensure accuracy with reasonable efficiency.

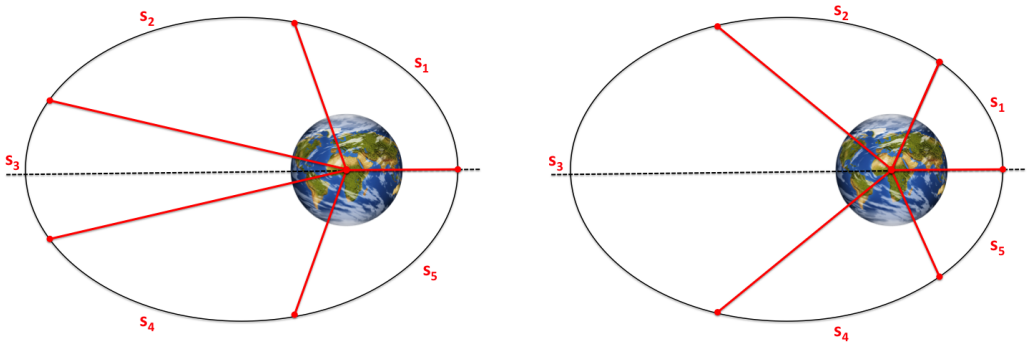
Relative Segment Length The segment length is the time period of interest for a single instance of MCPI, or equivalently the quantity $2w_2 = (t_f - t_0)$ from Figure 1.2. The segment length $2w_{2,j}$ is the length of the j^{th} segment s_j. A heuristic observation from the conservation of angular momentum of an unperturbed orbit is that evenly spaced timesteps around an orbit correspond to large physical displacements near perigee, and to smaller physical displace-

ments near apogee. This effect can be mitigated if evenly spaced steps are taken in true anomaly instead of time. This is shown graphically in Figure 3.1, which schematically illustrates the division of an orbit into five even length segments in time, and in true anomaly. If the five segments in the figure were MCPI intervals of approximation, then the true anomaly scheme in Figure 3.1b is preferred to the time sampling scheme in Figure 3.1a since it places MCPI nodes more densely in the most non-linear regions of the orbit near perigee, where the orbital velocity is the highest. This is especially true in light of the cosine density sampling of the MCPI segments at the CGL nodes, as shown in Figure 1.1. Near perigee, the perturbation forces due to the Earth (spherical harmonic gravity, atmospheric drag) will also be the most pronounced and rapidly varying, so having more MCPI nodes in these regions is expected to provide a better approximation to the true dynamics.

Order of Approximation The order of approximation of the segment s_j is denoted N_j, and is the order of the Chebyshev Polynomial series used to approximate the dynamics within the segment. Choosing the order of approximation N_j means we will have $N_j + 1$ cosine density sample points at the CGL nodes from Equation 1.6. As we can see in Figures 1.2 and 1.3, $N_j + 1$ function evaluations are required to perform each Picard iteration.

3.2.2 Heuristic Constraints on Tuning Parameters

Based upon examination of the true anomaly segmentation scheme in Figure 3.1, we shall specify several constraints on the MCPI tuning parameters. In preliminary studies, we found the good news that the minima associated with optimal tuning were locally very broad. This permits imposition of symmetry constraints to reduce the number of tuning parameters, which, in the case of the procedurally generated



(a) Five equal time segments. (b) Five equal true anomaly segments.

Figure 3.1: Schematic of moderately eccentric orbit divided into segments of equal length in time (Figure 3.1a) and true anomaly (Figure 3.1b).

tuning set, also serves to restrict the search space to heuristically reasonable values which helps the algorithm converge. Extensive experiments indicate that the run-time loss of performance caused by reducing the parameter space through symmetry constraints is less than 10%, and this loss can be largely recouped through a separate adaptive tuning process.

As a stronger constraint than specifying that there shall be $n \geq 1$ segments per orbit, we specify that the number of segments per orbit shall be an odd number $n = 1, 3, 5, ...,$ and that the first segment shall start at perigee (more generally, the point of closest approach on perturbed orbits) as shown in Figure 3.1b. This constraint is obviously relaxed on the first segment of a new orbit, since the initial conditions will occur at an arbitrary point in the orbit. Furthermore, we shall impose the constraint that symmetric segments (s_1 and s_5, s_2 and s_4, etc.) shall have the same length, and the same Chebyshev order of approximation. We have found that this pattern is near optimal and imposing odd symmetry greatly reduces the space of tuning parameters. This symmetry ensures that the maximum number of MCPI

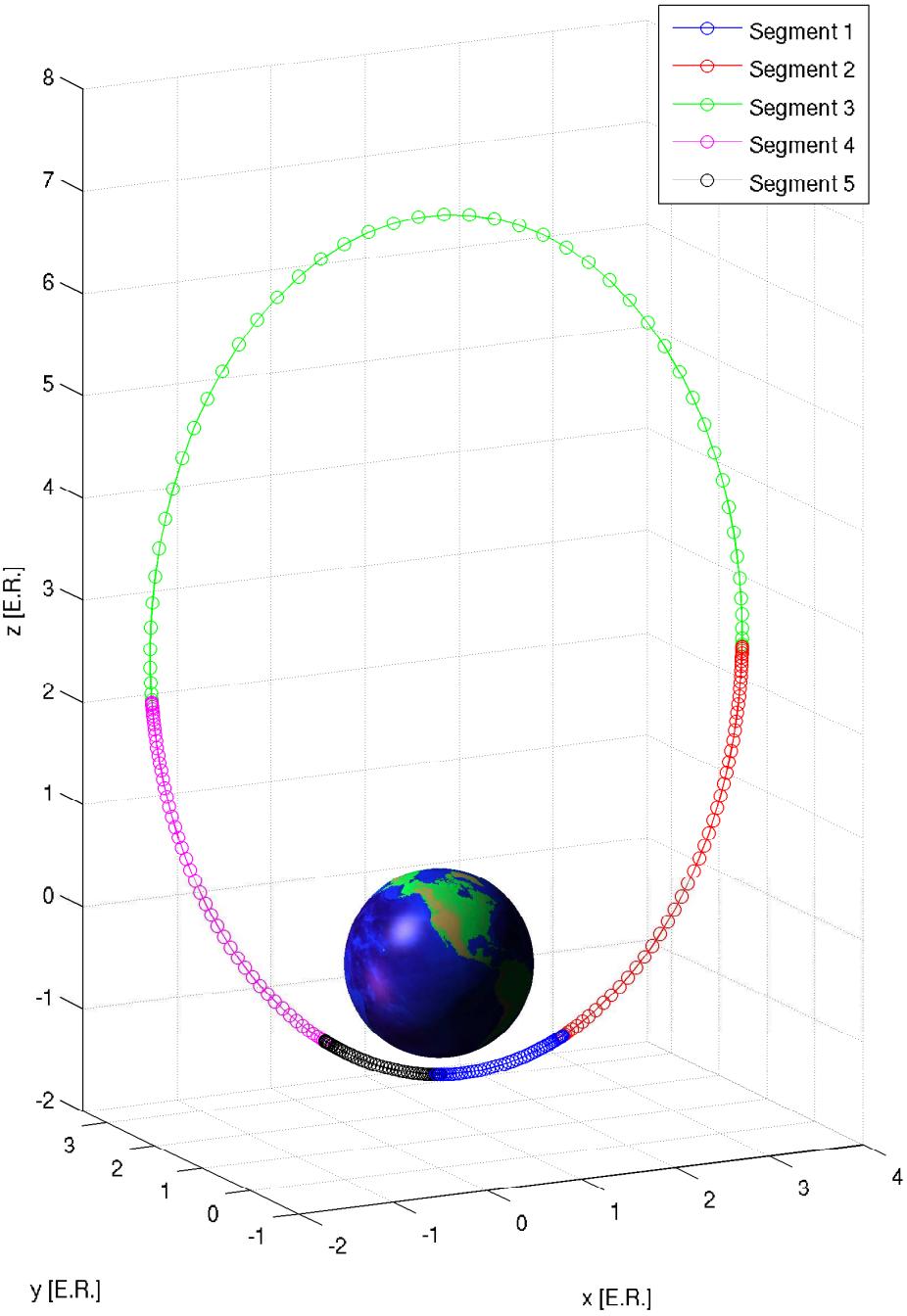


Figure 3.2: Hand-tuned MCPI segmentation for Molniya orbit propagation, designed to provide numerical precision Hamiltonian conservation.

nodes are near perigee where the acceleration changes most rapidly (since the first segment will begin there and the last segment will end there), and that the most spatially sparse nodes will be near apogee where the dynamics are changing the slowest (since the middle segment will cross apogee at its center). Additionally, this approach reduces the dimension of the parameter space by imposing heuristic lessons learned by hand tuning MCPI prior to developing this automated process, e.g. the Molniya orbit segmentation hand tuned to numerical precision that is shown in Figure 3.2. General ephemeris generation will begin from some epoch position within the orbit that is not at perigee, so the first MCPI segment end time will be adjusted in order to end on one of the "prescribed" symmetric segment endpoints. These prescribed points are in fact interpolated from a map generated by the one-time *a priori* parametric tuning process discussed below. The perturbation forces will cause the orbit elements to change, so the segment boundaries are re-calculated after each segment to correspond to the instantaneous (osculating) orbit, and each perigee passage point is determined such that $\dot{\boldsymbol{r}} = 0$ ($\boldsymbol{r}$ is a minimum) to initiate the subsequent orbit.

3.2.3 Performance Metrics

In order to compare one set of MCPI tuning parameters to another, we must define specific criteria over which to optimize. We would like to choose an appropriate set of MCPI parameters that minimizes the computational cost of the MCPI algorithm while constraining the resulting accuracy of computed ephemeris.

The computational cost of numerical orbit propagation is dominated by the evaluation of the force functions acting on the satellite. By far the most costly is the spherical harmonic gravity series. Therefore, by minimizing the number of gravity function evaluations over the space of tuning parameters, we effectively minimize the

computational cost of the MCPI algorithm. The gravitation potential field of the Earth at a particular radius, latitude, and longitude (r, ϕ, λ) is modeled as a spherical harmonic series, as shown in Equation 2.3. When implemented in code, the double summation in Equation 2.3 is a double `for` loop, and the upper limit of the outer summation is not infinity, but instead some L_{max}. Our MCPI implementation makes use of a radially adaptive spherical harmonic gravity algorithm that adjusts the complexity (size of the spherical harmonic gravity series calculated) dynamically such that a required accuracy is achieved with a reduced computational cost, as described in Section 2.2. If the radially adaptive gravity algorithm determines that at a given MCPI node the same accuracy can be achieved by calculating the series to degree and order $L < L_{max}$, then we compute an "equivalent gravity" computation cost that scales approximately as $E \propto \left(\frac{L}{L_{max}}\right)^2$, as given in Equation 2.7. We seek to minimize the total equivalent gravity function cost around the entire orbit, over the space of the tuning parameters considered.

There are many ways to quantify the solution accuracy of a numerical integration algorithm [62, 63]. For the purposes of the study in this dissertation, only spherical harmonic gravity perturbations are considered, however additional perturbation models can be added to the analysis. Because gravity is the only perturbation considered, the system is conservative and the total energy (Hamiltonian) preservation may be used as an accuracy metric. Although convenient, Hamiltonian preservation is an imperfect metric for completely describing numerical integrator performance because it does not capture all errors, and is known to be relatively insensitive to in-track errors for general orbit eccentricities and force models. For instance, on a circular orbit, all points have the same energy, and thus the Hamiltonian preservation metric is blind to the in-track errors [62]. More to the point, preservation of an energy integral can be considered necessary but not sufficient. We rely on

the much stronger contraction mapping nature of MCPI convergence to control the in-track error that may not be exposed by the Hamiltonian preservation metric. If, in the future, other non-conservative forces are added to the analysis then a different accuracy metric may be used instead, or else the total energy dissipated by the non-conservative forces may be integrated separately and the total work-energy conservation may be used. Furthermore, we can easily use the radially adaptive gravity model of Section 2.2 to truncate the spherical harmonic series consistent with the number of significant figures sought in numerically solving the orbital equations of motion.

For a given gravity representation, we choose to use two independent variables to describe the space over which we wish to find a set of MCPI tuning parameters: the perigee radius of the orbit, and the eccentricity of the orbit. The perigee radius is chosen because it approximately correlates to the strength and spatial granularity of the non-linearity (and timescale of variation) of the dynamics due to both spherical harmonic gravity and drag. The eccentricity describes the shape of the orbit, as well as the variation between the acceleration dynamics near perigee and the dynamics near apogee. If other perturbations are to be added in a later analysis that behave differently, for example the radiation pressure effect which is more sensitive to the orientation of the orbit than the shape, more independent variables can readily be added to create a higher-dimensional space.

3.2.4 Summary

Summarizing the above, the goal of the automated tuning parameter selection for orbit propagation with MCPI is as follows:

For a given:

1. Perigee radius

2. Eccentricity
3. Set of perturbation models
4. Required physical accuracy

Design a scheme to choose:

1. How many MCPI segments into which to divide the orbit (n)
2. The relative lengths of the segments in true anomaly ($w_{2,j}$)
3. The order of approximation of each segment (N_j)

Such that:

1. The required physical accuracy is achieved, approximately uniformly, all the way around the orbit
2. The MCPI node point distribution is optimally placed, densely near perigee where the dynamics are changing most rapidly in space and time, and sparsely near apogee where the acceleration is smaller and has less spatial variability
3. The total number of function evaluations around the orbit is minimized
4. The radial adaptive gravity is exploited such that the total number of "equivalent gravity" function evaluations (gravity computational cost) is minimized.

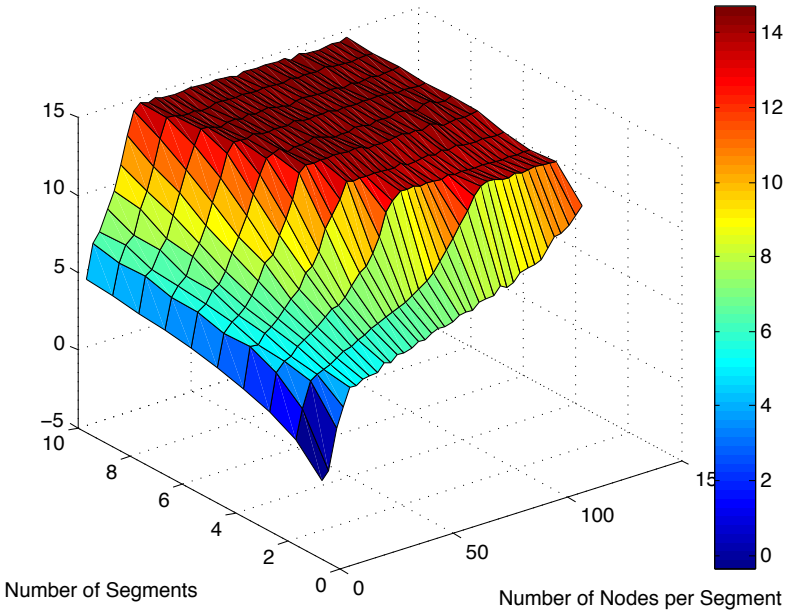
3.3 Empirically Generated Tuning Set

An MCPI tuning parameter scheme has been generated using an empirical "brute force" method. It provides a baseline tuning parameter set that is guaranteed to satisfy the hard accuracy constraint for an arbitrary orbit, however it does not rigorously

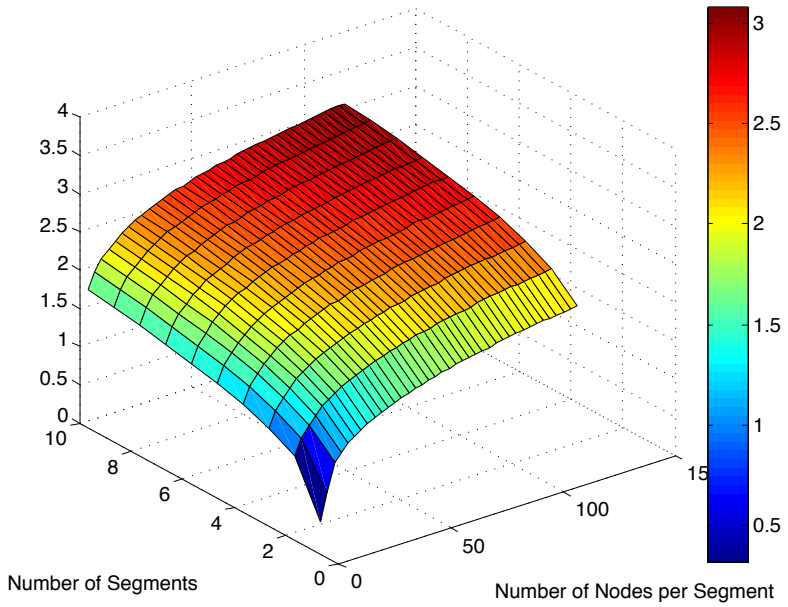
enforce the minimization of the number of force function calls (therefore optimal efficiency is not achieved). Essentially it provides a safe set of parameters, but is not claimed to be a computationally most efficient set. While sub-optimal, numerical studies indicate that only in rare circumstances will this parameter set result in greater than 20% more computational expense than the truly optimally tuned set.

The empirical parameter scheme is generated by choosing to use three MCPI segments of lengths (100°, 160°, 100°) in true anomaly around the orbit (the two 100° segments meet at perigee and the 160° segment is centered on apogee). These segment length boundaries were found by hand-tuning MCPI for a variety of orbits and choosing a value that delivered good results over a wide range of orbits. All segments have the same order of approximation, and a given physical accuracy requirement is chosen for the ephemeris solution. An orbital eccentricity tolerance is chosen ($e \geq 0.01$), and for orbits less eccentric than the tolerance (more circular), three segments of even length around the orbit are used instead of the empirically selected true anomaly segment breaks defined above.

The choice to use three segments is motivated by Figure 3.3 and the truth that, since prior studies have shown that extremely accurate solutions can be achieved with only one segment per orbit, we are confident that state of the art precision and reasonable efficiency can be achieved with three segments. Figure 3.3a shows the worst-case number of achievable digits of Hamiltonian conservation by MCPI when propagating an orbit using various numbers of segments (n) and numbers of nodes per segment (N). There is a large red plateau of numerical precision, and a steep slope up to the plateau. The plateau is not a physical limit, but rather a consequence of using double-precision arithmetic that limits the accuracy in the most precise subset of tuning parameter choices. 14+ digits of accuracy does not limit practical applications, since usually only 8 to 9 digits of accuracy are required for most near-



(a) Number of accurate significant digits of achievable worst-case Hamiltonian conservation. Red is good - higher achievable accuracy.



(b) $\log_{10}$ of the number of equivalent gravity function evaluations. Red is more expensive - higher gravity computation cost.

Figure 3.3: Parameter space for a Low Earth Orbit (eccentricity $e = 0.1$). Number of achievable digits of Hamiltonian conservation and equivalent gravity evaluation cost as a function of the number of (equal length) segments around the orbit, and the number of nodes per segment.

Earth applications, and 10 to 12 digits for solar applications. Figure 3.3b shows the $\log_{10}$ of the number of equivalent gravity evaluations required by MCPI, as a function of n and N. The red areas correspond to more computational cost and the blue areas to less. For any given Hamiltonian conservation requirement there is a finite region of parameter sets (n, N) that provide roughly equivalent gravity computation cost. The broad feasible set is good news, it means there is significant "forgiveness" for sub-optimal tuning. Because of the imposed symmetry constraint that n is odd, and the fact that $n = 1$ did not achieve the numerical accuracy plateau in this range of N, we choose to use 3 segments per orbit. Thus the only free parameter is the order of approximation N.

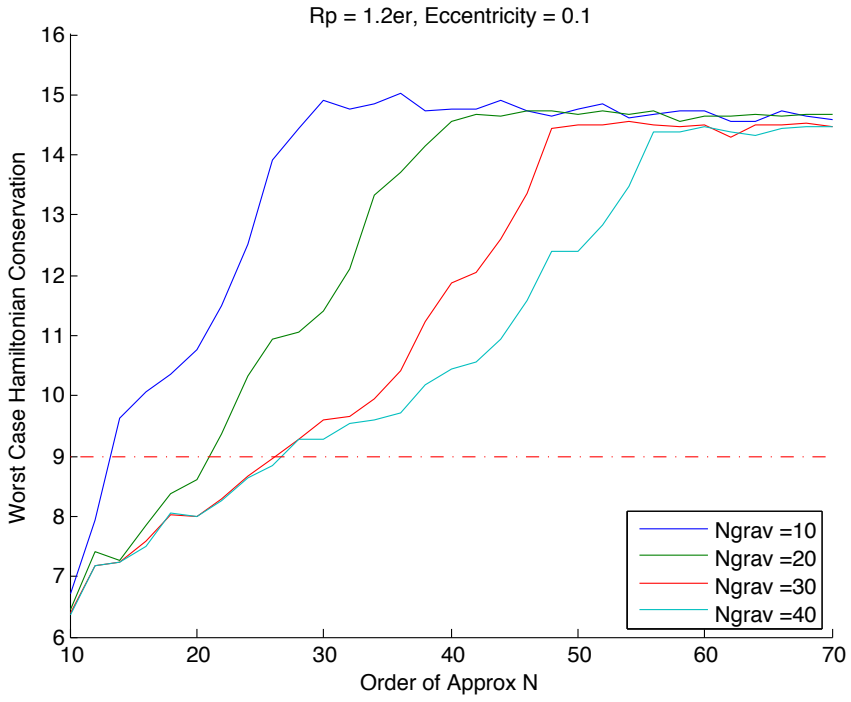


Figure 3.4: Worst case Hamiltonian conservation in segment (number of digits achieved), as a function of the number of MCPI nodes, for various spherical harmonic gravity field complexities.

To determine the required N for a single orbit (given a perigee radius and eccentricity), MCPI is used to propagate the first segment of the orbit starting from perigee. The order of MCPI approximation is varied, and the worst case Hamiltonian conservation is recorded. This process is repeated using a spherical harmonic gravity model of varying fidelity. The results of this process for a single representative case are shown in Figure 3.4. The desired physical accuracy requirement (number of desired digits of Hamiltonian preservation) is shown as the horizontal red dotted line in the figure. The required N to achieve the given accuracy is simply the horizontal location of the intersection of the curves with the dotted line. It is intuitively logical that using a higher complexity representation of the gravity field will require a higher order Chebyshev approximation, which is the behavior demonstrated in the figure.

This routine is then repeated for varying perigee radius and eccentricity in the feasible region of orbits in question. For each gravity field maximum degree and order, a parameter surface is generated. Figure 3.5 shows the surface of required MCPI order for various perigee radii and eccentricity in the presence of a 20x20 square spherical harmonic gravity field. Similarly, Figure 3.6 shows the surface in the presence of a 40x40 spherical harmonic gravity field. The full lookup table imported into MCPI consists of an array of surfaces like the ones shown, each one corresponding to a certain gravity field complexity. Note that, as the perigee radius increases, these surfaces become identical due to the fact that the high degree terms in the gravity model decay rapidly as orbital radius increases.

3.4 Genetic Algorithm Optimized Tuning Set

The above results lead to a "safe" set of segmentation break times and number of nodes that are designed to provide the required physical accuracy with near minimum execution cost. However choosing three fixed length segment breaks around the

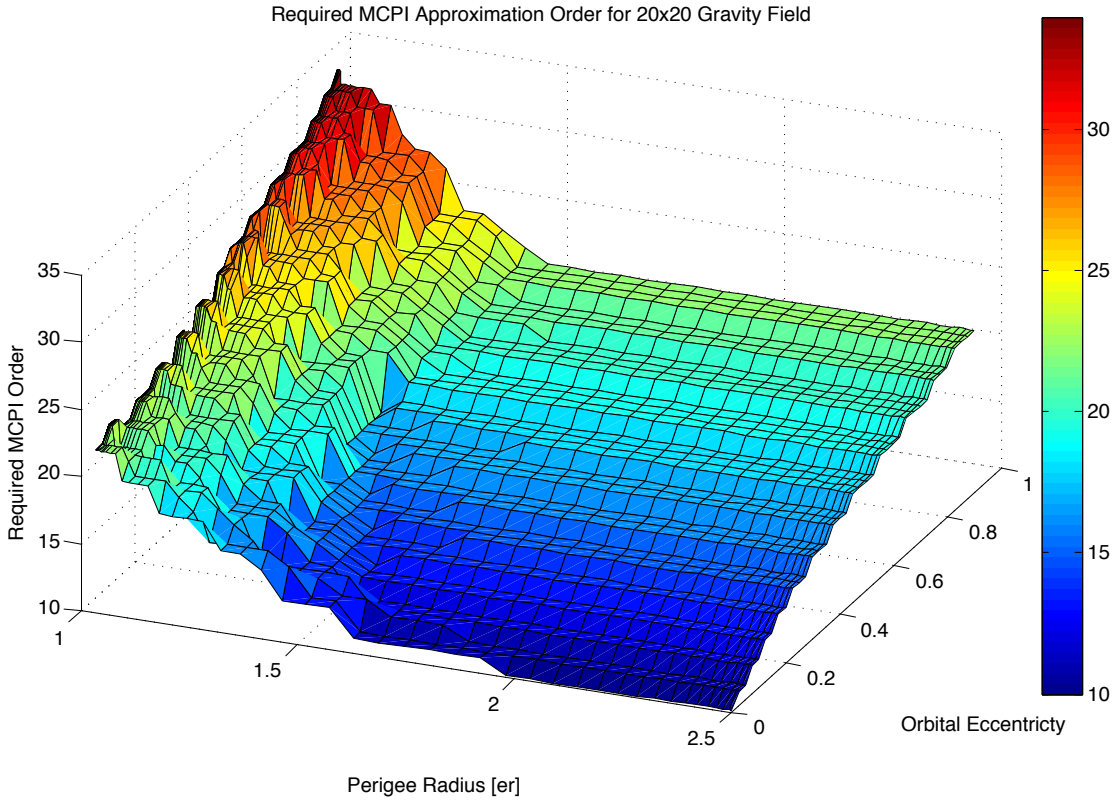


Figure 3.5: Empirically determined number of required MCPI nodes per segment in the presence of 20x20 spherical harmonic gravity perturbation, as a function of perigee radius and eccentricity.

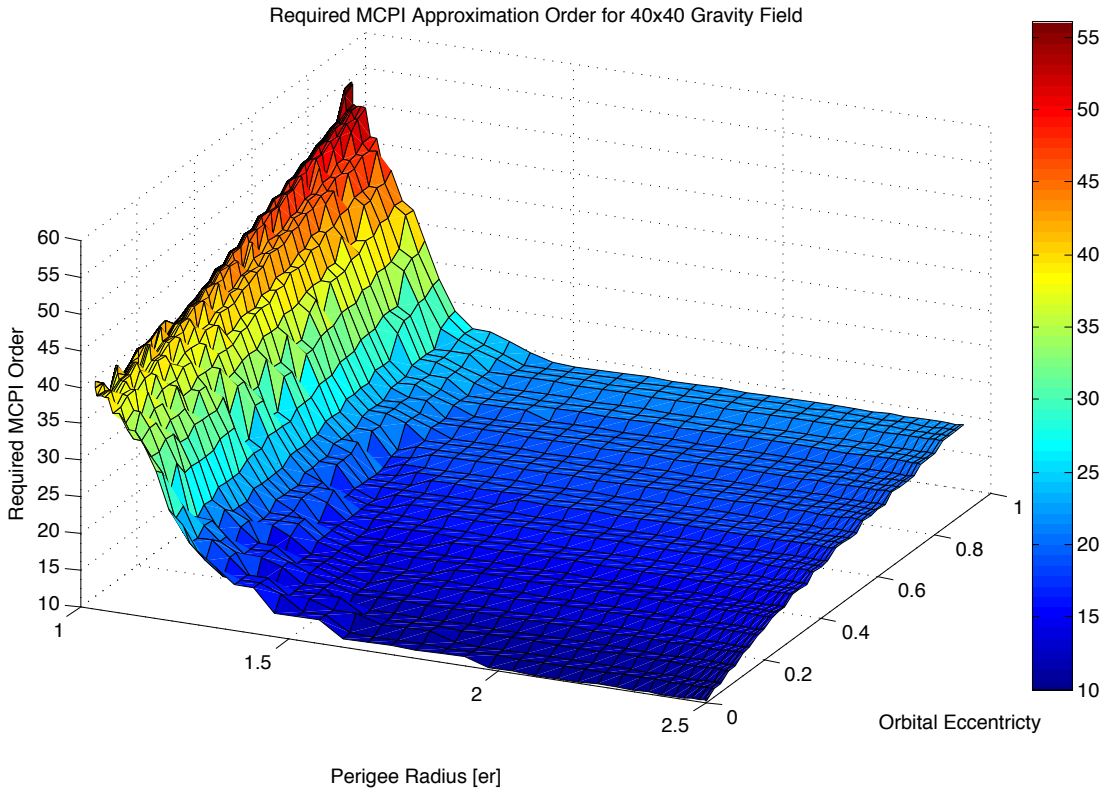


Figure 3.6: Empirically determined number of required MCPI nodes per segment in the presence of 40x40 spherical harmonic gravity perturbation, as a function of perigee radius and eccentricity.

orbit, and constraining all segments to the same order of approximation, guarantees that the parameter choices will necessarily be somewhat suboptimal. Promising initial results have been obtained using a global constrained optimization scheme (a genetic algorithm) on the problem at hand. Many approaches could be taken to do this optimization. A genetic algorithm was chosen because of its generality: It has no requirement of differentiability or convexity of the penalty function. It also allows for linear and non-linear constraints, parameter bounds, and integer optimization, all of which are required for this problem.

Consistent with the empirical approach, we will again constrain the number of segments to $n = 3$ around the orbit, with the first and third segments meeting at orbit perigee. The length of the segments $w_{2,j}$ and the order of approximation of the segments N_j is allowed to vary, but we maintain the constraint of symmetric segments, i.e. $w_{2,1} = w_{2,3}$ and $N_1 = N_3$. The length of the middle segment is related to the length of the first and third segments by the expression

$$w_{2,2} = W_r w_{2,1} = W_r w_{2,3} \tag{3.1}$$

where W_r is a proportionality factor. Similarly, the order of the middle segment is related to order of the first and third by

$$N_2 = N_r N_1 = N_r N_3 \tag{3.2}$$

where N_r is a second proportionality factor. Therefore the total number of nodes around the orbit from the three segments is $N_{tot} = 2N_1 + N_2 = N_1(2 + N_r)$. The proportionality constants and the number of nodes required in the first and third segment will vary from orbit to orbit, and we seek to determine them as a function

of the perigee radius r_p and eccentricity e:

$$N_1 = c_0 + (c_1 r_p + c_2 r_p e + c_3 e) \tag{3.3a}$$

$$N_r = 1 + (c_4 r_p + c_5 r_p e + c_6 e) \tag{3.3b}$$

$$W_r = 1 + (c_7 r_p + c_8 r_p e + c_9 e). \tag{3.3c}$$

We have introduced the $\boldsymbol{c}$ vector containing c_0 through c_9 which are coefficients on the linear and cross-coupling dependence of the perigee radius and eccentricity. The vector $\boldsymbol{c}$ is made up of the tuning parameters (called a gene in the parlance of the genetic algorithm) for which we seek to determine optimal values with the genetic algorithm.

A customized implementation of a genetic algorithm has been designed to maximize a "fitness" function over the space of genes[1]. It attempts to iteratively determine the fittest gene (parameter set) by "breeding" and "mutating" the most fit genes to replace the least fit genes. It starts with an initial "population" of genes, which can be randomized perturbations from a given protogene, or can be purely random. An iteration consists of: *(i)* evaluating the fitness of the current population, *(ii)* sorting the genes by their fitness values, *(iii)* keeping a subset of the most fit genes while replacing the more unfit genes by breeding together the more fit genes and randomly mutating the unfit genes. This implementation was designed to run in parallel on a compute cluster for efficiency, and can therefore handle larger population sizes than an earlier serial Matlab version (100 is a typical population size). The LASR SSA cluster was used to generate the results shown, allowing for 190 parallel fitness evaluations at a time. Specifications of the LASR SSA cluster are given in Appendix A.

[1]Credit is due to my colleague, Austin Probe. This work is a refined version of his initial implementation of the customized parallel genetic algorithm framework.

The fitness value F for a given gene is a summation around the orbit over the MCPI nodes k (from $k = 0$ to $k = N_{tot}$) of the multiplicative product of the two performance metrics discussed in Section 3.2.3 (namely: *(i)* the equivalent gravity cost, and *(ii)* the Hamiltonian preservation). This summation is subtracted from some arbitrarily high maximum possible fitness F_{max}, since we wish to maximize fitness. Therefore F is defined:

$$F = F_{max} - \sum_{k=0}^{N_{tot}} (E_k) \left(|\log_{10}(H_k) - \log_{10}(H_{desired})| + 1 \right) \tag{3.4}$$

where E_k is equivalent gravity function evaluation cost at the k^{th} node (as defined in Equation 2.6), H_k is the achieved Hamiltonian preservation value at the k^{th} node, and $H_{desired}$ is the desired value of Hamiltonian preservation around the orbit. The second multiplicative term in the fitness function $(|\log_{10}(H_k) - \log_{10}(H_{desired})| + 1)$ is a penalty weighted by the difference of the number of significant digits of achieved Hamiltonian preservation from the number of digits of desired Hamiltonian preservation. Uneven Hamiltonian preservation around the orbit, as shown in Figure 3.7, is punished by this term. The curve in Figure 3.7 is the value of H_k plotted over one orbit period for a single representative gene. The three segment boundaries are denoted by the black vertical dashed lines. The effect of the sub-optimal tuning parameter set is evident by the bimodal humps of Hamiltonian preservation of 10^{-9} for the two segments near perigee in the orbit, and the flat section near 10^{-11} for the segment passing apogee. For the case plotted, $H_{desired} = 10^{-9}$, and the accuracy requirement is met. However, this is a suboptimal gene because the Hamiltonian is not preserved evenly around the orbit. This means computational effort is being wasted, and that the boundaries of the segments should be adjusted to more evenly preserve the Hamiltonian all the way around the orbit. An optimal gene would have

flat Hamiltonian preservation all the way around the orbit, with the preservation magnitude being equal to $H_{desired}$. The Hamiltonian is a convenient metric to use in this study because only spherical harmonic gravity perturbations are considered, but in the case of general perturbations the analysis could be repeated with any preferred accuracy metric. In practice, after tuning we can preserve the Hamiltonian to almost graphical precision of the specified tolerance (down to about 10^{-14}), when using double precision arithmetic.

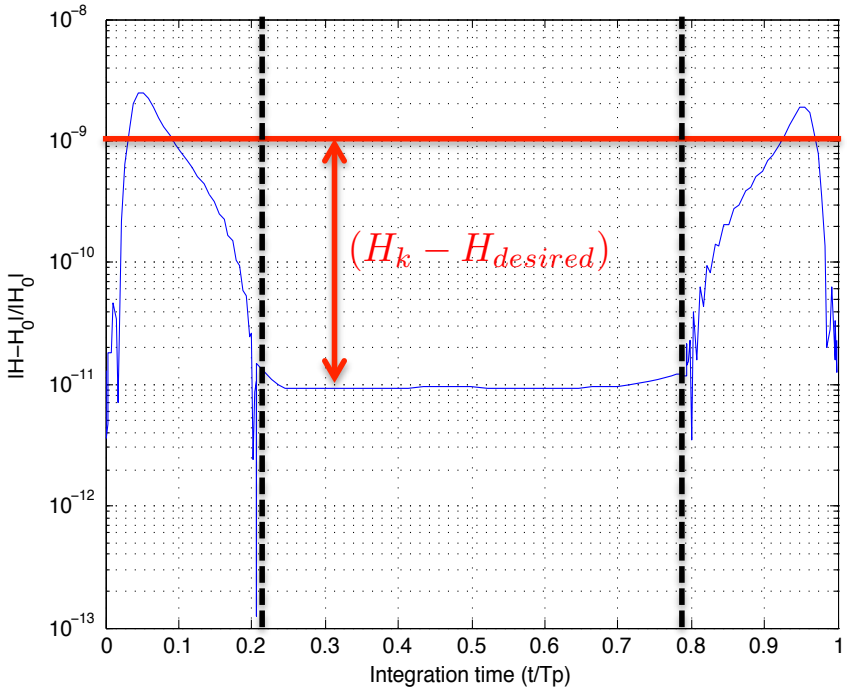


Figure 3.7: Normalized Hamiltonian conservation around orbit for a single gene within genetic algorithm, compared with desired Hamiltonian preservation value.

Within a single iteration of the genetic algorithm for a single gene, the fitness F is calculated for a large number of orbits within the orbital regime of interest (since

the algorithm runs on a cluster we can use a large number of different orbits - 500 is typical) and the overall fitness F_{total} is calculated. Using a large number of different orbits ensures that the fittest gene (tuning parameter set) is optimal over an ensemble average of representative orbits, and not just a single orbit or set of orbits. Current focus is to find optimal tuning parameter sets for general orbit catalog propagation, however if it were desired to create tuning parameter sets optimized for certain orbital regimes such as LEO or GEO, this could similarly be achieved.

3.5 Comparison of Tuning Sets

Figure 3.8 shows preliminary results from the genetic algorithm, starting from a purely randomized population. The blue curve is the maximum fitness value of the fittest gene, plotted by iteration number. The numerical fitness value itself is not meaningful, so the initial fitness value has been subtracted from all values to show the convergence trend. For comparison, the red line is the fitness value of the empirically determined tuning scheme, plotted with the initial fitness subtracted. The genetic algorithm is able to find a slightly better tuning parameter set than the heuristically motivated empirical scheme, after about 120 iterations. The genetic algorithm itself has several tuning parameters that can be optimized, which may lead to faster convergence and more optimal values. Additionally, different initial populations can be explored to ensure the entire parameter space is fully covered, which will also lead to better results and faster convergence. The fact that the genetic algorithm result is not drastically more optimal (by this optimality criterion) than the empirical method result seems to indicate that seeking further computational speedup with an *a priori* tuning set like these two methods will be subject to diminishing returns. Furthermore, empirical studies indicate that the optimal tuning lies in a broad feasible region with low curvature. This means, qualitatively, that the

performance of neighboring (in parameter space) sub-optimal parameter sets is near optimal.

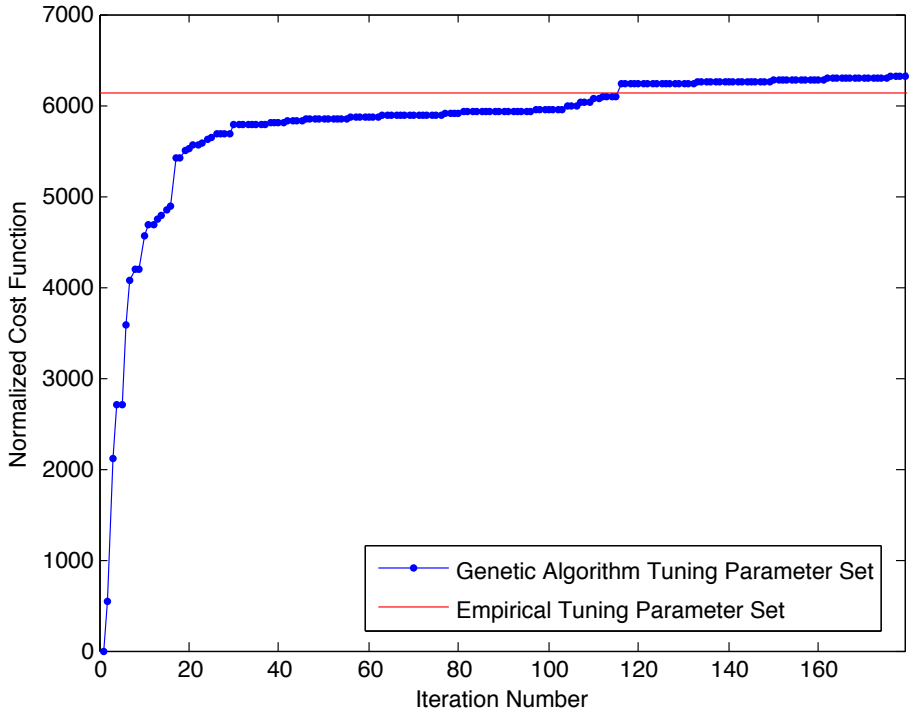


Figure 3.8: Normalized maximum fitness value of fittest gene, plotted with respect to iteration number.

These initial parameter set selection methodologies are designed to choose a reasonable set of tuning parameters based upon the properties of the orbit being propagated. The chosen parameters can be locally further adapted during the MCPI convergence process, as needed, in order to maintain the accuracy tolerance in the presence of other force functions, or after detection of the fact that the tolerance would not be met with the chosen parameter set. An adaptive algorithm can locally

vary the parameters from these *a priori* chosen parameter sets as it determines that variations are necessary. Whereas the two initial tuning parameter sets are particular to perturbed orbital motion, this adaptive algorithm is generally applicable to any ODE system. Indeed, rather than attempting to squeeze further optimality from the *a priori* tuning schemes, a process that quickly exhibits diminishing returns, further research effort will bear much more fruit by instead focusing on the development of this adaptive scheme. An outline for a locally adaptive MCPI algorithm is given in Section 3.6.

3.6 Adaptive Method for General Problem

The first order and second order MCPI algorithms described in Section 1 both use a least-squares Chebyshev approximation of the integrand as a first step of each Picard iteration. The number of CGL sample points M at which to fit the Chebyshev series for acceleration, velocity, and position has been (until now) chosen to be $M = N$, where N is the Chebyshev order of approximation of the position states. In the second order MCPI case the Chebyshev approximation order of the integrand is $N - 2$, since the order increases by one during the analytic integration process of the Chebyshev series, and second order MCPI is a double integrator method. Since the first order MCPI method is a single integrator, the Chebyshev approximation in that algorithm is of order $N - 1$. A reasonable value for the order of approximation N, and the segment length $t_f - t_0$ is chosen based upon the tuning parameter sets in the previous sections, and the integrand is fit using the Chebyshev least-squares method. This fit is the critical step which governs the ultimate success or failure of the MCPI algorithm to achieve the convergence tolerance for the chosen parameter set. Examining the integrand least squares fit residual errors provides an explicit measure of the health of future Picard iterations over the segment of approxima-

tion. If the integrand fit residuals of the initial integrand Chebyshev approximation, starting from a reasonable warm/hot start (see Section 2.4), are not within the the convergence threshold, the method is doomed to failure for that particular segment and choice of tuning parameters. At this point the segment length can be decreased, or the order of approximation increased, such that the method is likely to succeed. This leads to a simple to implement adaptation scheme. We adjust the order N to achieve a good fit unless $N > N_{max}$, in which case we reduce the segment length. Using the empirical *a priori* tuning parameter set based upon perigee radius, eccentricity, and desired accuracy, the adaptive process is seldom required to make large adjustments in N or the interval length.

4. CONSTRAINED MCPI

4.1 Introduction

This section describes a novel algorithm for application of MCPI to Ordinary Differential Equation systems that intrinsically have associated conserved quantities. MCPI may safely be applied to such systems without making use of this algorithm, but it is shown that the convergence rate may be accelerated, the domain of convergence may be expanded, or better ultimate solution accuracy for the same convergence threshold may be achieved by making use of the algorithm. Several examples of varying complexity explore the benefits of the new algorithm for constrained dynamical systems.

4.2 Background

Consider a general Ordinary Differential Equation (ODE)

$$\dot{\boldsymbol{z}} = \frac{d\boldsymbol{z}}{dt} = \boldsymbol{f}(t, \boldsymbol{z}(t)) \tag{4.1}$$

which is to be numerically solved using MCPI. This ODE is written in terms of an augmented state vector $\boldsymbol{z}$, and allows us to represent systems that are naturally first order systems as well as naturally second order systems. In the case of a naturally first order system $\boldsymbol{z} \equiv \boldsymbol{x}$, where $\boldsymbol{x}$ is the $(n \times 1)$ vector containing the system states. In the case of a naturally second order system, $\boldsymbol{z} \equiv [\boldsymbol{x}^T, \dot{\boldsymbol{x}}^T]^T$ is a $(2n \times 1)$ vector containing the system position states and the system velocity states, and the system has been implicitly demoted from a second order system to a first order system.

Let us consider ODE systems for which there are one or more associated *conserved*

quantities

$$h_r = h_r(\boldsymbol{p}, \boldsymbol{z}(t)) \equiv {}^0h_r \tag{4.2}$$

where h_r is the r^{th} such exact integral's numerical value, (optionally) $\boldsymbol{p}$ is a constant vector of parameters associated with the ODE system, and ${}^0h_r$ is the initial value of the quantity from the given boundary conditions of the problem. Note that there is an implicit assumption that the boundary conditions are consistent with the conserved quantities, such that ${}^0h_r$ is the true (valid) value of the exact integral. The true system dynamics evolve such that $h_r = {}^0h_r$ for all time. This is equivalent to stating that the constraint equation

$$g_r = g_r(\boldsymbol{p}, \boldsymbol{z}(t)) \equiv {}^0h_r - h_r(\boldsymbol{p}, \boldsymbol{z}(t)) \equiv 0 \tag{4.3}$$

must be true along the path of true system dynamics, for all time. While the differential equation implicitly (theoretically) conserves $h_r(\boldsymbol{p}, \boldsymbol{z}(t))$, due to finite numerical precision and other details of the numerical integration algorithm, $h_r(\boldsymbol{p}, \boldsymbol{z}(t))$ will not be conserved exactly. The process of numerical propagation will introduce small numerical inaccuracies into the solution such that $h_r \neq {}^0h_r$ in general.

Viewed independently, each constraint g_r defines an n-dimensional manifold (where n is the length of the $\boldsymbol{z}$ vector) on which the true system dynamics evolve. Combining all relevant constraints g_r, for $r = 1, 2, ..., R$, defines an intersection of manifolds upon which the true dynamics are constrained. Within this manifold intersection sub-space there are an infinite number of possible state-space trajectories, but only one which satisfies the system equations of motion given by Equation 4.1 (assuming that the differential equation initial value problem has a unique solution).

There exists a class of numerical integration methods called *symplectic integration*

methods. A consequence of the symplectic nature of this class of algorithms is that the Hamiltonian energy (or similar energy function) of the propagated state space solution is enforced by the numerical method for undamped systems of equations. Comparing MCPI to symplectic methods, we find that MCPI implicitly conserves the Hamiltonian energy as well as the symplectic methods because MCPI is a contraction mapping to the true system dynamics, and during terminal convergence approaches the true system dynamics to within machine precision. This section describes a method for using conserved system quantities, of which the Hamiltonian energy is one example, to accelerate MCPI in the convergence process and eliminate slow drift over long time intervals. Small constraint rectification corrections are solved for which correct the slight departures of the state trajectory estimate back onto the intersection of state-space manifolds (in which the true solution is constrained to lie). However this process is much more general than that of a symplectic integrator, which implicitly conserves Hamiltonian energy. This process may be applied simultaneously to any number of general conserved quantities of the system. By "telling" the intermediate Picard iteration state estimates about constraint errors, we are able to accelerate convergence and further stabilize the long term error drift.

4.3 Theory

4.3.1 Minimum Correction Constraint Restoration for First-Order ODEs

In the process of numerically solving the ODE with MCPI, we have some approximation of the augmented state vector $\boldsymbol{z}$ over the MCPI segment length. During the i^{th} MCPI iteration, the augmented state matrix Z^i contains the i^{th} estimate of the

system states along the MCPI segment of approximation

$$Z^i = \begin{bmatrix} z_1^i(\tau_0) & z_2^i(\tau_0) & \dots & z_n^i(\tau_0) \\ z_1^i(\tau_1) & z_2^i(\tau_1) & \dots & z_n^i(\tau_1) \\ \vdots & \vdots & \vdots & \vdots \\ z_1^i(\tau_N) & z_2^i(\tau_N) & \dots & z_n^i(\tau_N) \end{bmatrix} \tag{4.4}$$

in normalized time ($-1 \leq \tau \leq 1$), evaluated at the $M+1$ CGL nodes (as defined in Equation 1.7). At the k^{th} timestep τ_k, evaluation of the r^{th} system conserved quantity will not yield identically zero, but rather some small constraint violation

$$g_r(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k)) = {}^0h_r - h_r(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k)) \equiv \epsilon_r(\tau_k) \tag{4.5}$$

Working in the realm of step-by-step numerical integrators, Nacozy proposed a correction for the value of the state at the next timestep $\boldsymbol{y} = \boldsymbol{z} + \boldsymbol{\Delta z}$ such that the constraint error is eliminated ($g_r(\boldsymbol{p}, \boldsymbol{y}) = 0$). He demonstrated a constrained minimization framework to choose the smallest $|\boldsymbol{\Delta z}|$ with a single first-order correction, such that the constraint equation is satisfied [64]. MCPI does not work in a timestep-by-timestep fashion, but instead approximates long arcs of the system state trajectories at each stage of the computation. However, a similar constrained minimization approach is applicable, but *over all timesteps at once*.

Picard iteration is theoretically attracted to the exact solution. To preserve this attractive feature we do not modify Picard iteration itself, but rather nullify small arithmetic errors in the state estimates using a "constraint restoration" correction step. Picard iteration may then be resumed after each constraint restoration step, with the next iteration beginning at the corrected state estimate, which now lies more precisely on the constraint manifold intersection. We find, upon implementing

this notion, that the final Picard iterations usually remain on the constraint manifold intersection to high precision, and thus the constraint restoration steps may be considered the ultimate "hot-start" for subsequent Picard iterations. While the constraint restoration corrects the state estimates back to the nearest point on the intersection of constraint manifolds, the final Picard iteration updates the state estimates such that the differential equation is satisfied. In some sense, this approach is safer than Nacozy's step-by-step method because the final step is always a Picard iteration, which is theoretically attracted to the exact solution of the differential equation. This claim can not be made for the Explicit Runge-Kutta algorithms (for which Nacozy designed his method).

The Minimum Correction Constraint Restoration (MCCR) process is now described. For each of the $M+1$ CGL function evaluation points k, we desire a state correction factor $\boldsymbol{\Delta z}_k$ such that the instantaneous constraint error $\epsilon_r(\tau_k)$ vanishes

$$g_r(\boldsymbol{p}, \boldsymbol{z}(\tau_k) + \boldsymbol{\Delta z}_k) \to 0 \tag{4.6}$$

Note that the state vector $\boldsymbol{z}$ is of length n, and there are $(M+1)$ CGL nodes at which we seek a correction, so we desire to find $n(M+1)$ total individual correction terms. Expanding the constraint equation at a single CGL node at time τ_k gives

$$\begin{aligned} g_r(\boldsymbol{p}, \boldsymbol{z}(\tau_k) + \boldsymbol{\Delta z}_k) = {}^0h_r - h_r(\boldsymbol{p}, \boldsymbol{z}(\tau_k)) - \eta_r(\boldsymbol{p}, \boldsymbol{z}(\tau_k), \boldsymbol{\Delta z}(\tau_k)) + \\ + O(\boldsymbol{\Delta z}(\tau_k)^2) + ... \equiv 0 \end{aligned} \tag{4.7}$$

where $\eta_r(\boldsymbol{p}, \boldsymbol{z}(\tau_k), \boldsymbol{\Delta z}(\tau_k))$ is a bi-linear expression in terms of the states $\boldsymbol{z}(\tau_k)$ and the correction factors $\boldsymbol{\Delta z}(\tau_k)$. Similar to Nacozy's approach we will use a first-order correction, therefore neglecting all factors of order $\boldsymbol{\Delta z}(\tau_k)^2$ and higher. This is a reasonable approximation to make because the constraint error terms ϵ_r have

been found to be small when MCPI has nearly converged to the true solution, to within small arithmetic errors. The term ${}^0h_r - h_r(\boldsymbol{p}, \boldsymbol{z}(\tau_k))$ in Equation 4.7 is simply the original expression for the instantaneous constraint error $\epsilon_r(\tau_k)$ (as given by Equation 4.5). Substituting these changes in Equation 4.7 gives

$$\eta_r(\boldsymbol{p}, \boldsymbol{z}(\tau_k), \boldsymbol{\Delta z}(\tau_k)) = \epsilon_r(\tau_k) \tag{4.8}$$

which relates a bi-linear expression in the instantaneous states and state corrections to the instantaneous constraint error at the time τ_k. Evaluating this expression at each of the $M+1$ CGL function evaluation points τ_k and stacking the results yields a matrix equation

$$H_r \boldsymbol{\Delta Z} = \boldsymbol{\epsilon_r} \tag{4.9}$$

where $\boldsymbol{\epsilon_r}$ is an $(M+1)$ vector containing all the small constraint errors at the $M+1$ timesteps, and $\boldsymbol{\Delta Z}$ is an $n(M+1)$ vector containing all the state corrections at all timesteps:

$$\boldsymbol{\Delta Z} = [\boldsymbol{\Delta z}_0(\tau_0), ..., \boldsymbol{\Delta z}_n(\tau_0), \boldsymbol{\Delta z}_0(\tau_1), ..., \boldsymbol{\Delta z}_n(\tau_1), \boldsymbol{\Delta z}_0(\tau_N), ..., \boldsymbol{\Delta z}_n(\tau_N)]^T \tag{4.10}$$

The matrix H_r is of size $(M+1) \times n(M+1)$, and contains the factors of the instantaneous state terms $\boldsymbol{z}(\tau_k)$ that manifested in the bi-linear η_r terms.

The formulation of Equations 4.5 through 4.10 is for the r^{th} constraint equation g_r. In general, an ODE system is able to have more than one constraint ($r = 1, 2, ..., R$). In the case of more than one constraint, Equation 4.9 may be augmented to handle all R constraints at once. To do so, the individual H_r matrices corresponding to each

constraint are stacked

$$H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_R \end{bmatrix} \tag{4.11}$$

to form a single $R(M+1) \times n(M+1)$ matrix. Similarly, the individual constraint error vectors $\boldsymbol{\epsilon_r}$ are stacked

$$\boldsymbol{\epsilon} = \begin{bmatrix} \boldsymbol{\epsilon}_1 \\ \boldsymbol{\epsilon}_2 \\ \vdots \\ \boldsymbol{\epsilon}_R \end{bmatrix} \tag{4.12}$$

to form a single $R(M+1)$ vector. The number of state corrections that we seek does not change, so the $\boldsymbol{\Delta Z}$ vector remains the same as defined in Equation 4.10. This new augmented system of equations may be written as simply

$$H\boldsymbol{\Delta Z} = \boldsymbol{\epsilon} \tag{4.13}$$

In practice, each H_r matrix will have a roughly block-diagonal structure due to the bi-linear nature of the η_r terms. Rather than augmenting the system by simply stacking the matrices as in Equations 4.11 and 4.12, it is more computationally efficient to interleave the equations stemming from each η_r within the augmented H matrix and ϵ vector, thus maintaining an overall block diagonal structure.

The way the augmented state vector $\boldsymbol{z}$ in Equation 4.1 is defined means that the order of the ODE system is always equal to the length n of the state vector. An n^{th} order ODE system has (at most) $n-1$ conserved quantities (motion constants) [65]. This means that the matrix system in Equation 4.13 will always be an underde-

termined system (has more unknowns than equations), and thus has an infinity of solutions. A convenient criteria to find a solution among the infinite possible set is to choose the one which minimizes the total correction magnitude $(\boldsymbol{\Delta Z})^T(\boldsymbol{\Delta Z})$ while satisfying Equation 4.13, a classical minimum norm problem. This is accomplished by minimizing the augmented scalar cost function

$$J = \frac{1}{2}(\boldsymbol{\Delta Z})^T(\boldsymbol{\Delta Z}) + \boldsymbol{\lambda}^T(\boldsymbol{\epsilon} - H\boldsymbol{\Delta Z}) \tag{4.14}$$

where $\boldsymbol{\lambda}$ is a vector of Lagrange multipliers. The extrema of this cost function occurs when the gradient with respect to the state corrections and the Lagrange multipliers is zero:

$$\nabla_{\Delta Z}(J) = \boldsymbol{\Delta Z} - H^T\boldsymbol{\lambda} = \mathbf{0} \tag{4.15a}$$

$$\nabla_{\lambda}(J) = \boldsymbol{\epsilon} - H\boldsymbol{\Delta Z} = \mathbf{0} \tag{4.15b}$$

The system in Equations 4.15 may be solved to yield the classical minimum norm solution [53] for constraint restoration

$$\boldsymbol{\Delta Z} = H^T(HH^T)^{-1}\boldsymbol{\epsilon} \tag{4.16}$$

Upon applying the correction, we find that the subsequent Picard iterations, especially the terminal convergence iterations, represent very nearly lateral displacements on the intersection of the constraint manifolds (and the final constraint conservation is better, even though the final step is a Picard iteration). Applying this approach in a multi-segment (long-term) propagation has been found to significantly improve (the already very good) stability and accuracy of MCPI. It also leads to more efficient convergence, and convergence over longer time intervals. We mention, in the case

that the classical minimum norm solution is poorly conditioned, we can alternatively make use of the singular-value decomposition (SVD) algorithm. In Matlab, the SVD minimum norm correction can be computed by $\boldsymbol{\Delta Z} = pinv(H)\boldsymbol{\epsilon}$.

4.3.2 Minimum Correction Constraint Restoration for Second-Order ODEs

In the case of a naturally second order ODE system

$$\ddot{\boldsymbol{x}} = \frac{d^2\boldsymbol{x}}{dt^2} = \boldsymbol{f}(t, \boldsymbol{z}(t)) \tag{4.17}$$

$\boldsymbol{z} \equiv [\boldsymbol{x}^T, \dot{\boldsymbol{x}}^T]^T$ is a $(2n \times 1)$ augmented state vector containing the system position states and the system velocity states. The time derivative of the velocity states is the right-hand-side of the ODE in Equation 4.17, and the time derivative of the position states is simply the velocity states. Thus there is an explicit kinematic relationship between the states. When using the second order MCPI formulation described in Section 1.4.4, the MCPI algorithm is "aware" of this kinematic relationship, and enforces it explicitly during each Picard iteration. In the previous section for a first order ODE system there is not any such kinematic relationship, therefore we are free to independently vary all of the states in the state vector with a small state correction $\boldsymbol{\Delta Z}$, as shown in Equation 4.10. In the case of a second order ODE, state corrections at each timestep $\boldsymbol{\Delta z_k} \equiv [\boldsymbol{\Delta x}(\tau_k)^T, \boldsymbol{\Delta \dot{x}}(\tau_k)^T]^T$ must be chosen such that they respect the explicit kinematic relationship between the states, a relationship which second order MCPI "expects" to exist.

Examining the second order MCPI vector/matrix update steps in Equations 1.52a through 1.54, it can be seen that the Chebyshev approximation coefficients for the

acceleration, velocity, and position states are

$$F^{i-1} = {}^{F}T^{T} V G(X^{i-1}, V^{i-1}) \tag{4.18a}$$

$$\beta^{i} \quad = V_0 + {}^{\beta}C_{\gamma} F^{i-1} \tag{4.18b}$$

$$\alpha^{i} \quad = X_0 + {}^{\alpha}C_{\gamma} \beta^{i} \tag{4.18c}$$

The above equations lead to a Picard update to the velocity and position states of the form

$$V^{i} = {}^{v}C_{x} \beta^{i} \tag{4.19a}$$

$$X^{i} = {}^{x}C_{x} \alpha^{i} \tag{4.19b}$$

The i^{th} estimate of the position and velocity states are written as a matrix by stacking the state values for all $M+1$ CGL sampled times τ_0 to τ_M as

$$X^{i} = matrix\{\boldsymbol{x}^{i}(\tau_0)^{T} \ ; \ \boldsymbol{x}^{i}(\tau_1)^{T} \ ; \ ... \ ; \ \boldsymbol{x}^{i}(\tau_M)^{T}\} \tag{4.20a}$$

$$V^{i} = matrix\{\dot{\boldsymbol{x}}^{i}(\tau_0)^{T} \ ; \ \dot{\boldsymbol{x}}^{i}(\tau_1)^{T} \ ; \ ... \ ; \ \dot{\boldsymbol{x}}^{i}(\tau_M)^{T}\} \tag{4.20b}$$

where each $\boldsymbol{x}^{i}(\tau_j) = [x_1^i(\tau_j) \ , \ x_2^i(\tau_j) \ , \ ... \ , \ x_n^i(\tau_j)]^T$ and $\dot{\boldsymbol{x}}^{i}(\tau_j) = [\dot{x}_1^i(\tau_j) \ , \ \dot{x}_2^i(\tau_j) \ , \ ... \ , \ \dot{x}_n^i(\tau_j)]^T$ is an $(n \times 1)$ vector.

Combining Equations 4.18 and 4.19 leads to expressions for the state updates in terms of the acceleration coefficients F^{i-1} and the constant matrices:

$$V^{i} = {}^{v}C_{x} \left\{ V_0 + {}^{\beta}C_{\gamma} F^{i-1} \right\} \tag{4.21a}$$

$$X^{i} = {}^{x}C_{x} \left\{ X_0 + {}^{\alpha}C_{\gamma} \left[V_0 + {}^{\beta}C_{\gamma} F^{i-1} \right] \right\} \tag{4.21b}$$

Equations 4.21 are the kinematic constraints relating the position and velocity states to the acceleration, written in terms of the acceleration Chebyshev coefficients. The second order MCCR method seeks small corrections ΔF to the acceleration coefficients such that the system constraints (Equation 4.7) are satisfied due to the corresponding state corrections at each timestep $\boldsymbol{\Delta z_k} \equiv [\boldsymbol{\Delta x}(\tau_k)^T, \boldsymbol{\Delta \dot{x}}(\tau_k)^T]^T$, caused by the kinematic constraints (Equations 4.21) evaluated at $(F^{i-1} + \Delta F)$. Partial derivatives of Equations 4.21 with respect to F^{i-1} yields the sensitivities of the velocity and position states to small changes in the acceleration coefficients:

$$\frac{\partial V^i}{\partial F^{i-1}} = {}^{v}C_{x}{}^{\beta}C_{\gamma} \tag{4.22a}$$

$$\frac{\partial X^i}{\partial F^{i-1}} = {}^{x}C_{x}{}^{\alpha}C_{\gamma}{}^{\beta}C_{\gamma} \tag{4.22b}$$

Therefore, the approximate variation of the velocity and position states as a function of the variation of the acceleration coefficients may be expressed as

$$\Delta V^i = {}^{v}C_{x}{}^{\beta}C_{\gamma}\Delta F \tag{4.23a}$$

$$\Delta X^i = {}^{x}C_{x}{}^{\alpha}C_{\gamma}{}^{\beta}C_{\gamma}\Delta F \tag{4.23b}$$

The integrand coefficient matrix F^{i-1}, and therefore the integrand coefficient correction matrix ΔF, is of size $(N-1) \times n$, where N is the Chebyshev order of approximation of the position states, and n is the size of the state vector $\boldsymbol{x}(t)$. The velocity and position matrices V^i and X^i, and subsequently the velocity and position correction matrices ΔV^i and ΔX^i, are of size $(M+1) \times n$, where $M = N$ is the number of CGL sample points. Therefore, the two matrices ${}^{v}C_{x}{}^{\beta}C_{\gamma}$ and ${}^{x}C_{x}{}^{\alpha}C_{\gamma}{}^{\beta}C_{\gamma}$

are size $(M+1) \times (N-1)$. For ease of notation, let us rewrite Equations 4.23 as

$$\Delta V^i = \Gamma \Delta F \tag{4.24a}$$

$$\Delta X^i = \Lambda \Delta F \tag{4.24b}$$

by simply renaming the matrices $\Gamma \equiv {}^{v}C_{x}{}^{\beta}C_{\gamma}$, and $\Lambda \equiv {}^{x}C_{x}{}^{\alpha}C_{\gamma}{}^{\beta}C_{\gamma}$. By inspection of Equations 4.20 and 4.24 it is evident that the correction to the velocity and position states at timestamp τ_k is simply

$$\boldsymbol{\Delta \dot{x}}(\tau_k) = \Gamma_k \Delta F \tag{4.25a}$$

$$\boldsymbol{\Delta x}(\tau_k) = \Lambda_k \Delta F \tag{4.25b}$$

for $k = 0, 1, 2, ..., M$, where Γ_k and Λ_k is shorthand for the $(k)^{th}$ row of the matrices Γ and Λ. This is taking a liberty with the matrix indexing notation which typically begins at 1, but we will begin numbering with 0 in this case.

As in Equation 4.6 of the first order MCCR method, we seek a state correction factor $\boldsymbol{\Delta z_k}$ at each timestep τ_k to cause the instantaneous constraint error $\epsilon_r(\tau_k)$ to disappear. However, we are not free to individually vary the position or velocity state correction terms $\boldsymbol{\Delta z_k} \equiv [\boldsymbol{\Delta x}(\tau_k)^T, \boldsymbol{\Delta \dot{x}}(\tau_k)^T]^T$, but instead seek corrections to the integrand coefficients ΔF, which affect the position and velocity states at timestamp τ_k through Equations 4.25. By the same procedure as Equation 4.7 of the first order constrained method, we expand the constraint equation in powers of $\boldsymbol{\Delta z_k}$ and keep terms of order one. Then, Equations 4.25 are substituted in to replace the state correction terms $\boldsymbol{\Delta \dot{x}}(\tau_k)$ and $\boldsymbol{\Delta x}(\tau_k)$ such that we end up with a bi-linear expression in terms of ΔF, and $\boldsymbol{z}(\tau_k)$ (and also the constant factors Γ_k and Λ_k). As

in Equation 4.9, we stack the bi-linear expressions to form a matrix equation

$$H_r \Delta F = \boldsymbol{\epsilon_r} \tag{4.26}$$

where the H_r matrix contains terms of the states $\boldsymbol{z}(\tau_k)$ and the elements of the constant matrices Γ and Λ. As in the first order method, we may again stack each of the matrix systems from the $r = 1, 2, ..., R$ constraints into an augmented system

$$H \Delta F = \boldsymbol{\epsilon} \tag{4.27}$$

Depending upon the number of constraints R, and the size of the state vector n, Equation 4.27 for the MCCR solution may be solved as a minimum norm problem as in Equation 4.16, as a simple matrix inverse

$$\Delta F = H^{-1} \boldsymbol{\epsilon} \tag{4.28}$$

or as a least squares problem [53]

$$\Delta F = (H^T H)^{-1} H^T \boldsymbol{\epsilon} \tag{4.29}$$

Having solved for the integrand coefficient corrections terms, we may apply a constraint restoration correction to the values of the states V^i and X^i by substituting $F = F^{i-1} + \Delta F$ into Equation 4.18a and propagating the changes through Equations 4.18 and 4.19.

4.3.3 Discussion

A first order constraint restoration correction (first order in the sense that linearized expansion terms are utilized - not to be confused with the separate MCCR

methods for first and second order ODEs) has been found that, when applied to the value of the states at each timestep τ_k as $\boldsymbol{z}(\tau_k) + \boldsymbol{\Delta z}_k$, will cause the constraint error vector $\boldsymbol{\epsilon} \rightarrow \mathbf{0}$. This correction methodology works along the entire MCPI segment at once, in contrast to Nacozy's step-by-step method which works for a single timestep at a time [64]. The slight modification to the previous MCPI iteration to nullify the small constraint errors provides a "hotter" start for the next MCPI iteration. There are several ways to implement this idea. This segment correction method may either be used after the standard MCPI has converged as an *a posteriori* correction step for the final one or two MCPI iterations, or to provide small constraint restoration corrections within the MCPI convergence loop (as described above), correcting the MCPI state estimate slightly after every Picard iteration.

The effectiveness of using the method as an *a posteriori* correction depends upon the MCPI convergence tolerance chosen by the user. In the case that a non-strict convergence tolerance is being used, the *a posteriori* correction may provide a more accurate final solution. However, if a very strict MCPI convergence threshold is being used, then MCPI will have already solved for a very accurate estimate of the true solution (Picard iteration is theoretically a contraction mapping of the state estimate to the true dynamics) and the precision achievable in the matrix inverse of Equation 4.16 will be the limiting factor for additional solution accuracy. An additional factor for consideration is if some conserved quantity is desired to be used as the metric for assessing numerical integrator solution accuracy, for instance in Section 3 when the Hamiltonian preservation is used as a metric for choosing tuning parameters. Using the same conserved quantity as an *a posteriori* correction could cause the propagator performance (solution accuracy) to be overestimated (i.e. the actual solution accuracy is not as good as the MCCR value of the conserved quantity would make it seem). However, this problem will not arise when using the

correction in the MCPI loop, as the very slightly corrected state estimate near the terminal convergence is effectively a "hot-start" for the subsequent (and also the final) Picard iteration. We must keep in mind that there are generally an infinity of paths on a given constraint manifold intersection surface, but only one of them will be the true solution. Prudence indicates that we should rely on the final contraction mapping character of MCPI to approximate the actual solution well, accelerated by the MCCR's small corrections to nullify the constraint residuals due to arithmetic errors on intermediate iterations.

Using the MCCR method following each iteration within the MCPI convergence loop is the preferred mode of operation. It is not without cost, as the matrix system of Equation 4.16 must be solved each time. However, as will be shown in the following examples, the benefits of using the method far outweigh the small computational cost. In the initial few MCPI iterations, when the MCPI convergence errors are larger, and thus the nodal correction factors could be larger, it is sometimes necessary to apply the constraint restoration correction more than once per Picard iteration. This is because we have chosen to use a first-order correction, under the assumption that the constraint errors $\epsilon_r(\tau_k)$ are small. Applying it twice allows for the first-order correction to work, even if the corrections are not small. Towards the end of the MCPI convergence process, the solution does not vary much from iteration to iteration, and thus the small error assumption is typically well satisfied. Note that any *small MCCR linearization errors do not accumulate* because the final Picard iterations can converge from an infinity of approximations neighboring the true solution. The validity of these remarks is easily demonstrated using a few specific examples, which are provided in Section 4.4.

4.4 Examples

4.4.1 First-Order Example: Torque-Free Rigid Body Motion

The Euler rotational equations of motion

$$[I]\dot{\boldsymbol{\omega}} = -\boldsymbol{\omega} \times ([I]\boldsymbol{\omega}) + \boldsymbol{L_c} \tag{4.30}$$

are a set of coupled, non-linear, first-order differential equations that describe the evolution of the angular velocity vector $\boldsymbol{\omega} = [\omega_1, \omega_2, \omega_3]^T$ of a rigid body with body-fixed moment of inertia matrix $[I]$, subject to external torque $\boldsymbol{L_c}$. Assuming that there are no external torques acting upon the body ($\boldsymbol{L_c} \equiv \boldsymbol{0}$), and that the body-fixed coordinate frame is judiciously chosen along the body principle axes, such that the moment of inertia matrix is diagonal, the Euler equations reduce to the simpler form

$$\dot{\omega}_1 = -\frac{(I_{33} - I_{22})}{I_{11}}\omega_2\omega_3 \tag{4.31a}$$

$$\dot{\omega}_2 = -\frac{(I_{11} - I_{33})}{I_{22}}\omega_3\omega_1 \tag{4.31b}$$

$$\dot{\omega}_3 = -\frac{(I_{22} - I_{11})}{I_{33}}\omega_1\omega_2 \tag{4.31c}$$

starting from some given initial condition $\boldsymbol{\omega}(t_0) = [\omega_1(t_0), \omega_2(t_0), \omega_3(t_0)]^T$ at time t_0. In the notation of Equation 4.1, the state vector $\boldsymbol{z} \equiv \boldsymbol{\omega}$ is of length $n = 3$, and the system parameter vector $\boldsymbol{p}$ contains the moment of inertia matrix diagonal terms $\boldsymbol{p} = [I_{11}, I_{22}, I_{33}]^T$. From here onward, the inertia matrix diagonal terms will be written as I_1, I_2, and I_3 for notational simplicity.

Due to the absence of external torques, this ODE has two motion constants. The

system kinetic energy is constrained to lie upon the surface of the energy ellipsoid

$$T = \frac{1}{2}I_1\omega_1^2 + \frac{1}{2}I_2\omega_2^2 + \frac{1}{2}I_3\omega_3^2 \tag{4.32}$$

and the magnitude of the system angular momentum vector $\boldsymbol{H}$ is conserved. Equivalently it may be said that the system is constrained to lie upon the momentum ellipsoid

$$H^2 = \boldsymbol{H}^T\boldsymbol{H} = I_1^2\omega_1^2 + I_2^2\omega_2^2 + I_3^2\omega_3^2 \tag{4.33}$$

The energy ellipsoid and the momentum ellipsoid describe two manifolds in state space, and it is along the intersection of these two manifolds that the true system dynamics must evolve in time [41]. The constancy of the system angular momentum magnitude and kinetic energy means that these quantities should always be equal to their initial values at time t_0 (at the given initial boundary conditions), as in Equation 4.2:

$$h_1 \equiv T = {}^0T \tag{4.34a}$$

$$h_2 \equiv H^2 = {}^0H^2 \tag{4.34b}$$

Equivalently, the above equations may be restated as constraint equations as in Equation 4.3:

$$g_1 \equiv {}^0T - T \equiv 0 \tag{4.35a}$$

$$g_2 \equiv {}^0H^2 - H^2 \equiv 0 \tag{4.35b}$$

During the process of numerically solving the ODE system with MCPI, we will have some best approximation of the system states along the MCPI segment of

interest. During the i^{th} iteration, at the k^{th} of $(M+1)$ CGL sampled timesteps τ_k, the current best estimate of the state is $\boldsymbol{z}^i(\tau_k)$. The constraints of Equation 4.35 will not be exactly satisfied. Instead, the constraint equations will exhibit some small constraint violations $\epsilon_1(\tau_k)$ and $\epsilon_2(\tau_k)$:

$$g_1(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k)) = {}^0T - T(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k)) \equiv \epsilon_1(\tau_k) \neq 0 \tag{4.36a}$$

$$g_2(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k)) = {}^0H^2 - H^2(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k)) \equiv \epsilon_2(\tau_k) \neq 0 \tag{4.36b}$$

We seek a set of constraint restoration state corrections $\boldsymbol{\Delta z}(\tau_k)$ that will serve to drive the small instantaneous constraint violation to zero

$$g_1(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k) + \boldsymbol{\Delta z}(\tau_k)) \to 0 \tag{4.37a}$$

$$g_2(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k) + \boldsymbol{\Delta z}(\tau_k)) \to 0 \tag{4.37b}$$

Expanding the constraint equation $g_1(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k)+\boldsymbol{\Delta z}(\tau_k))$ as in Equation 4.7 yields

$$\begin{aligned} g_1(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k) + \boldsymbol{\Delta z}(\tau_k)) = 0 = {}^0T - \frac{1}{2} I_1 [z_1(\tau_k) + \Delta z_1(\tau_k)]^2 \\ - \frac{1}{2} I_2 [z_2(\tau_k) + \Delta z_2(\tau_k)]^2 - \frac{1}{2} I_3 [z_3(\tau_k) + \Delta z_3(\tau_k)]^2 \end{aligned} \tag{4.38}$$

Recognizing the constraint violation term $\epsilon_1(\tau_k)$ from Equation 4.36a the substitution

$${}^0T - \frac{1}{2} I_1 z_1(\tau_k)^2 - \frac{1}{2} I_2 z_2(\tau_k)^2 - \frac{1}{2} I_3 z_3(\tau_k)^2 = \epsilon_1(\tau_k) \tag{4.39}$$

may be made. Doing so, and neglecting the terms of $O(\Delta z(\tau_k)^2)$ leaves the simple expression

$$\epsilon_1(\tau_k) = I_1 z_1(\tau_k) \Delta z_1(\tau_k) + I_2 z_2(\tau_k) \Delta z_2(\tau_k) + I_3 z_3(\tau_k) \Delta z_3(\tau_k) \tag{4.40}$$

which is bi-linear in the state $\boldsymbol{z}$ and the state corrections $\boldsymbol{\Delta z}$, as in Equation 4.8.

Similarly, expansion of the second constraint equation $g_2(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k)+\boldsymbol{\Delta z}(\tau_k))$ gives

$$g_2(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k) + \boldsymbol{\Delta z}(\tau_k)) = 0 = {}^0H^2 - I_1^2[z_1(\tau_k) + \Delta z_1(\tau_k)]^2 \\ - I_2^2[z_2(\tau_k) + \Delta z_2(\tau_k)]^2 - I_3^2[z_3(\tau_k) + \Delta z_3(\tau_k)]^2 \tag{4.41}$$

Neglecting terms of $O(\Delta z(\tau_k)^2)$ and substituting the constraint violation term $\epsilon_2(\tau_k)$ as

$${}^0H^2 - I_1^2 z_1(\tau_k)^2 - I_2^2 z_2(\tau_k)^2 - I_3^2 z_3(\tau_k)^2 = \epsilon_2(\tau_k) \tag{4.42}$$

leaves the bi-linear expression

$$\frac{1}{2}\epsilon_2(\tau_k) = I_1^2 z_1(\tau_k)\Delta z_1(\tau_k) + I_2^2 z_2(\tau_k)\Delta z_2(\tau_k) + I_3^2 z_3(\tau_k)\Delta z_3(\tau_k) \tag{4.43}$$

Evaluating the bi-linear expressions of Equations 4.40 and 4.43 at each of the $(M + 1)$ CGL sampled times τ_k and collecting the terms into a matrix system, gives two matrix equations of the form of Equation 4.9. In order to calculate a set of corrections that enforces both constraints at once, the two matrix equations need to be augmented together to form a single matrix system. This may be done in a straightforward manner by stacking the two equations, as in Equations 4.11 through 4.13. However, since both matrix systems have a block-diagonal structure, a more computationally efficient method of joining the two matrix systems is by interleaving the two constraints from each sample point τ_k. This allows the overall block diagonal structure to be maintained, meaning sparse matrix algorithms or block inverse properties may be utilized for computational efficiency. The interleaved augmented matrix structure is written as:

$$\begin{bmatrix} I_1 z_1(\tau_0) & I_2 z_2(\tau_0) & I_3 z_3(\tau_0) & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ I_1^2 z_1(\tau_0) & I_2^2 z_2(\tau_0) & I_3^2 z_3(\tau_0) & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & I_1 z_1(\tau_1) & I_2 z_2(\tau_1) & I_3 z_3(\tau_1) & \dots & 0 & 0 & 0 \\ 0 & 0 & 0 & I_1^2 z_1(\tau_1) & I_2^2 z_2(\tau_1) & I_3^2 z_3(\tau_1) & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & I_1 z_1(\tau_M) & I_2 z_2(\tau_M) & I_3 z_3(\tau_M) \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & I_1^2 z_1(\tau_M) & I_2^2 z_2(\tau_M) & I_3^2 z_3(\tau_M) \end{bmatrix} \begin{bmatrix} \Delta z_1(\tau_0) \\ \Delta z_2(\tau_0) \\ \Delta z_3(\tau_0) \\ \Delta z_1(\tau_1) \\ \Delta z_2(\tau_1) \\ \Delta z_3(\tau_1) \\ \vdots \\ \Delta z_1(\tau_M) \\ \Delta z_2(\tau_M) \\ \Delta z_3(\tau_M) \end{bmatrix} = \begin{bmatrix} \epsilon_1(\tau_0) \\ \frac{1}{2}\epsilon_2(\tau_0) \\ \epsilon_1(\tau_1) \\ \frac{1}{2}\epsilon_2(\tau_1) \\ \vdots \\ \epsilon_1(\tau_M) \\ \frac{1}{2}\epsilon_2(\tau_M) \end{bmatrix} \tag{4.44}$$

Equation 4.44 may be written much more compactly as Equation 4.13. The H matrix is a $2(M+1) \times 3(M+1)$, the $\boldsymbol{\Delta Z}$ vector is of length $3(M+1)$, and the $\boldsymbol{\epsilon}$ vector is of length $2(M+1)$. This system may be solved for the state correction vector $\boldsymbol{\Delta Z}$ using the minimum norm formulation in Equations 4.14 through 4.16.

Choosing numerical values for the constants of the problem at hand, the benefit of this constrained MCPI methodology is explored. The diagonal moment of inertia matrix is set as $I = diag\{1.1, 2.1, 3.1\}$, and the initial condition of the angular velocity vector is chosen as $\boldsymbol{\omega}(t_0) = [0.01, 0.15, 0.2]^T$. Forward propagating the system states from the initial condition at $t_0 = 0$ to a later time $t_f = 14$s, the system states evolve as shown in the plot of Figure 4.1. MCPI was used to numerically integrate the dynamic system given by Equations 4.31, using an MCPI order of approximation of $N = 50$, and a convergence threshold of 10^{-12}. As an additional method of comparison, the solution for torque-free motion described by Equations 4.31 can be computed analytically in terms of the Jacobi elliptic functions [66], although that analysis is not provided here.

As discussed in Section 4.3.3, the constrained MCPI formulation may be utilized as either an *a posteriori* correction to the system states after MCPI has finished converging in an unconstrained manner, or it may be used during the MCPI convergence process, making a correction to the states every Picard iteration. Both of these methods are compared against unconstrained MCPI, and the resulting constraint violation is shown in Figure 4.2. The top plot shows the constraint violation term $\epsilon_1(t)$ due to the kinetic energy constraint of Equation 4.39, and the bottom plot shows the angular momentum constraint violation $\epsilon_2(t)$, as defined in Equation 4.42. All three methods are able reduce the constraint violation to the achievable double precision numerical error, indicating the numerical solution is a very highly accurate estimate of the true system dynamics. While minimal accuracy improvement is evident in

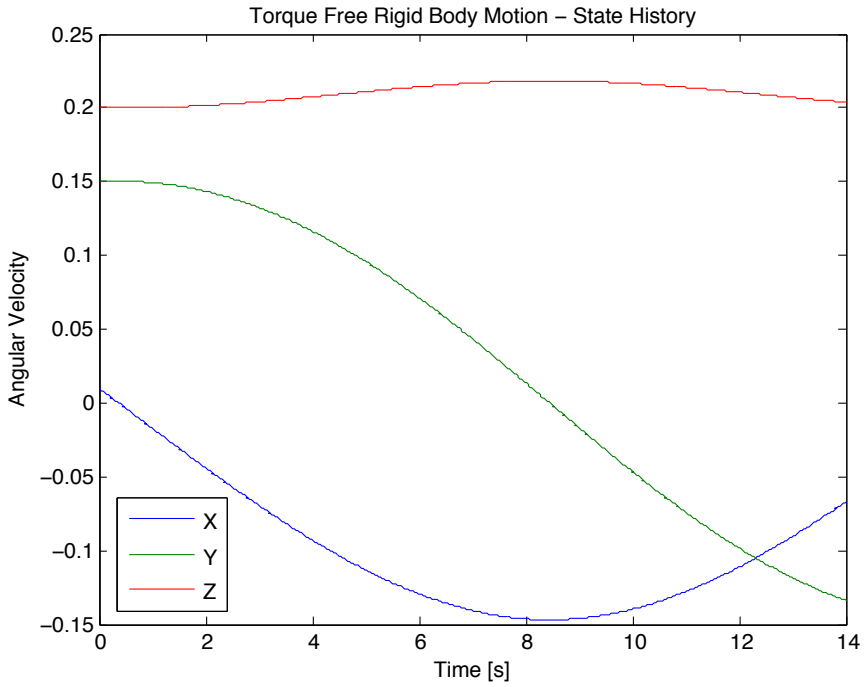


Figure 4.1: Torque-free rigid body motion, evolution of system states for $t_f = 14$s.

the results of Figure 4.2, these small improvements grow if hundreds of solutions are daisy-chained to obtain long-term motion prediction. More dramatically, however, it is perhaps surprising that making the constraint restoration corrections drastically accelerates the early (much lower accuracy) Picard iterations, as shown below.

The benefit of the constraint restoration methodology becomes clear when examining the convergence trend of this torque-free rigid body problem for MCPI with and without the constraint enforcement. Figure 4.3 shows the normalized MCPI convergence error, plotted versus iteration number, for constrained MCPI and unconstrained MCPI. In the unconstrained case, the algorithm converges in 25 iterations. The constrained algorithm converges in 14 iterations. The unconstrained case exhibits typical MCPI behavior whereby there are several initial iterations where the normalized error does not decrease much, although examination of the system states

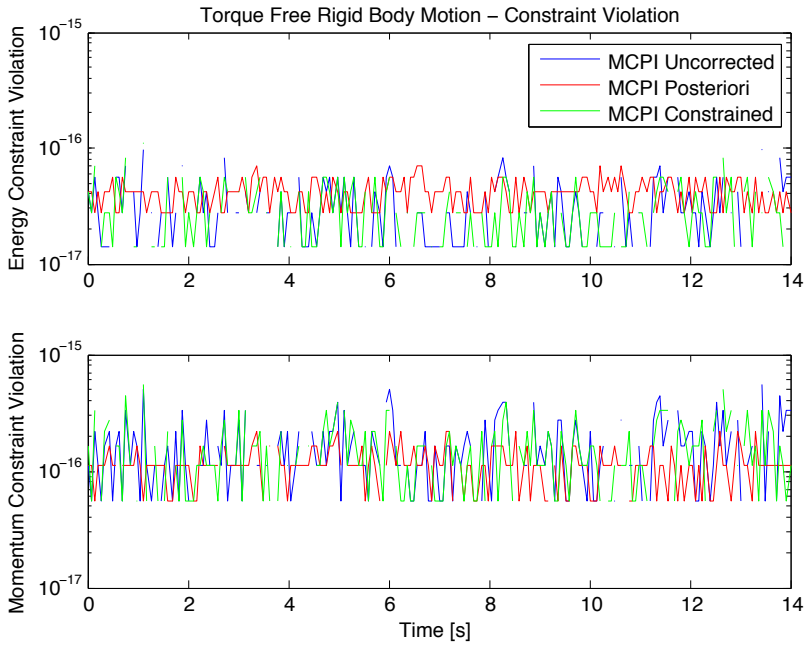


Figure 4.2: Torque-free rigid body motion, constraint violation for $t_f = 14$s using unconstrained MCPI, a single *a posteriori* correction, and constrained MCPI.

in state space does show that the approximation of the system states is improving every iteration. Providing the additional information implicit in the constraint restoration after each early (less accurately converged) Picard iteration shows that the corrected trajectory is much more accurate and accelerates the next MCPI corrected trajectory. After the initial iterations (5 or 6 of them in the unconstrained case), the MCPI geometric convergence behavior begins, where the convergence error decreases by nearly an order of magnitude every iteration. In the unconstrained case, MCPI converges 15 orders of magnitude in about 20 iterations. The constrained convergence trend decreases the number of initial (relatively) unproductive iterations by about half, and then vastly improves the rate of geometric convergence. The constrained MCPI algorithm converges 15 order of magnitude in roughly 11 iterations, or greater than an order of magnitude per iteration.

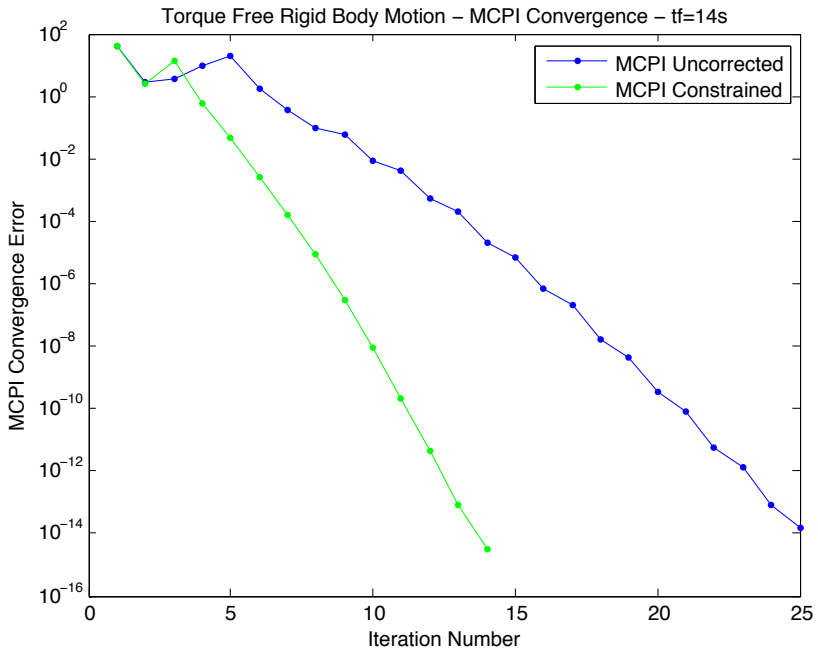


Figure 4.3: Torque-free rigid body motion, MCPI convergence trends for $t_f = 14$s using unconstrained MCPI versus constrained MCPI.

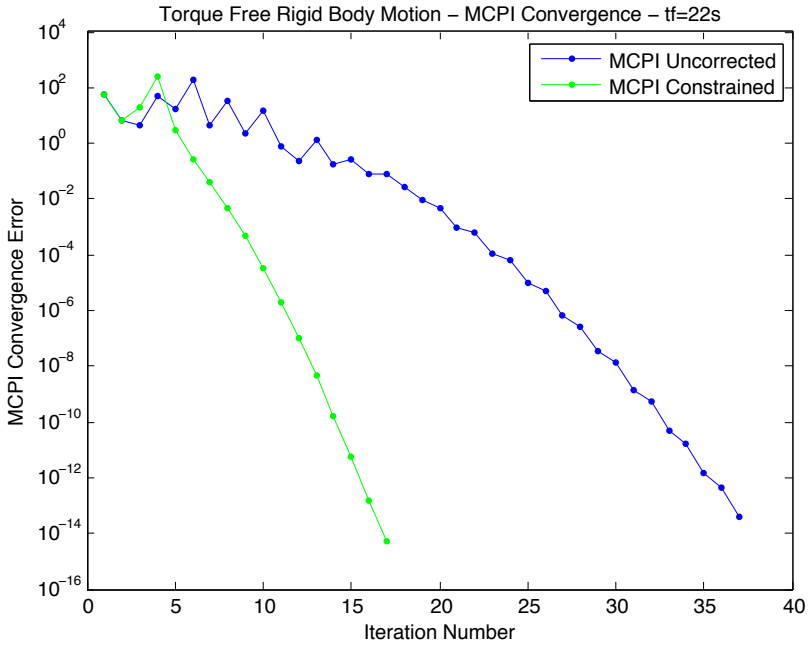


Figure 4.4: Torque-free rigid body motion, MCPI convergence trends for $t_f = 22$s using unconstrained MCPI versus constrained MCPI.

Increasing the MCPI segment length from $t_f = 14$s to $t_f = 22$s (still using an order of approximation $N = 50$), yields the convergence trends shown in Figure 4.4. In this case, unconstrained MCPI converges in 37 iterations, and the constrained MCPI method converges in 17 iterations. Thus the constrained method has decreased the required number of iterations by more than a factor of two. A similar trend is visible as in the previous case, where the constrained method has fewer initial iterations before achieving geometric convergence than the unconstrained case, and then a much steeper geometric convergence rate.

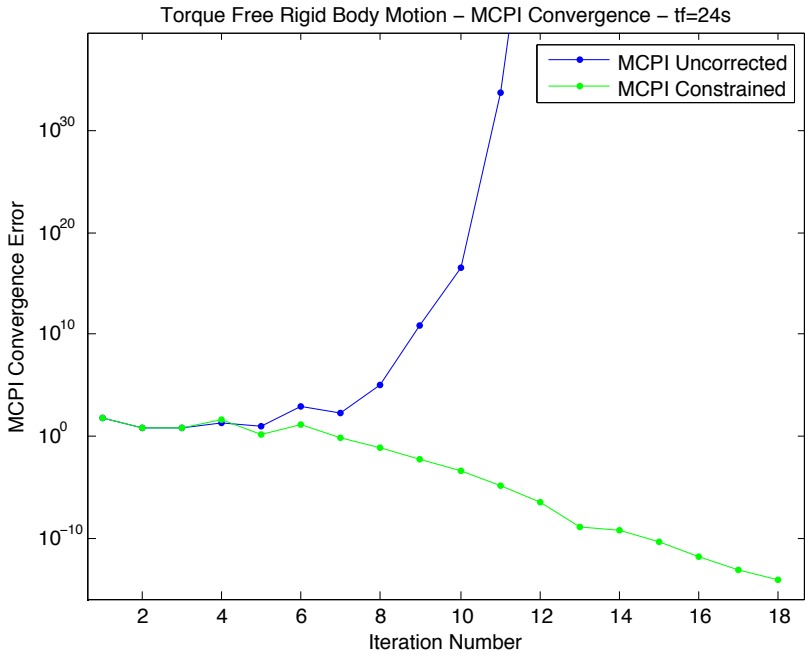


Figure 4.5: Torque-free rigid body motion, MCPI convergence trends for $t_f = 24$s using unconstrained MCPI versus constrained MCPI.

Further increasing the MCPI segment length to $t_f = 24$s, as shown in Figure 4.5, an interesting phenomenon occurs. The unconstrained MCPI is not able to converge

for this segment length and order of approximation, but the constrained MCPI is still well behaved and converges almost as efficiently as for the shorter segment cases. As indicated by the exponentially diverging blue curve, the unconstrained MCPI convergence error blows up to infinity, whereas the green constrained MCPI convergence curve shows the same geometric convergence rate as in previous cases. Apparently, using the improvements in the constraint restoration correction keeps the MCPI starting estimates for the Picard iterations within the smaller convergence domain of attraction for the larger time intervals.

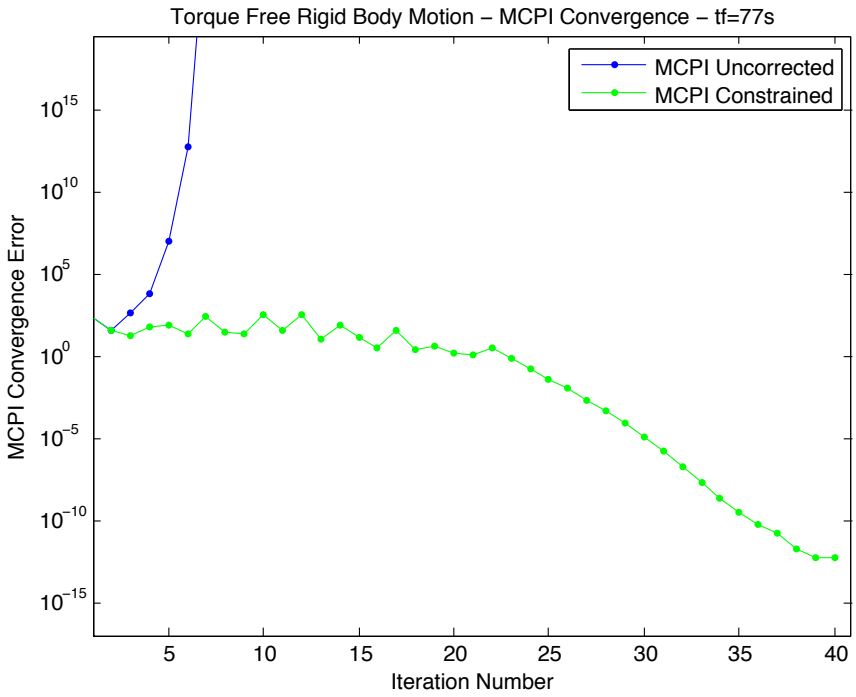


Figure 4.6: Torque-free rigid body motion, MCPI convergence trends for $t_f = 77$s using unconstrained MCPI versus constrained MCPI.

To further explore this phenomenon, the segment length is increased until the constrained MCPI algorithm is no longer able to converge. Empirically, this is found

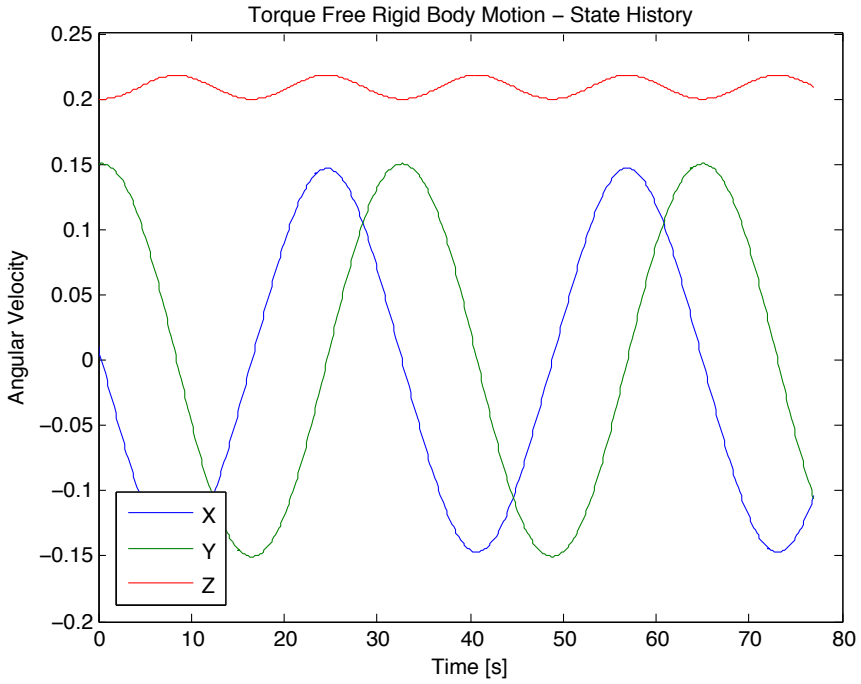


Figure 4.7: Torque-free rigid body motion, evolution of system states for $t_f = 77$s.

to be a segment length of $t_f > 77$s. Figure 4.6 shows the convergence trend of constrained MCPI, and the divergent behavior of unconstrained MCPI for $t_f = 77$s. The MCPI order of approximation is set to $N = 100$ for this case (because of the much longer time interval). Additionally, in the first few constrained MCPI iterations (when the normalized convergence error is ≥ 1) the state correction process is repeated twice due to the non-truth of the assumption that the state corrections $\boldsymbol{\Delta z}(\tau_k)$ are small, as discussed in Section 4.3.3. Thus, the constrained MCPI method is able to increase the MCPI domain of convergence from a segment length of $t_f = 23$s to a length of $t_f = 77$s, a 3.3X increase. The state output of the $t_f = 77$s case is shown in Figure 4.7.

In summary, the first order constrained MCPI method applied to the problem of torque-free rigid body motion has two-fold benefits. It is able to reduce the num-

ber of MCPI iterations in cases when the unconstrained method is able to converge, reducing the required number of iterations by more than half in some cases. Additionally, it is able to expand the convergence domain of the MCPI method more than three-fold.

4.4.2 Second-Order Example: Simple Harmonic Motion

A simple harmonic oscillator

$$\ddot{x} = -kx \ , \quad x(t_0) = x_0 \ , \quad \dot{x}(t_0) = v_0 \tag{4.45}$$

is an idealized, scalar, second order ODE system. In the notation of Equation 4.17, the state vector is $\boldsymbol{z} \equiv [x, \dot{x}]^T$ and therefore $n = 1$. The parameter vector simply contains the spring constant $\boldsymbol{p} \equiv [k]$. Conveniently, this system has an analytic solution

$$x(t) = c_1 \cos(\omega t) + c_2 \sin(\omega t) \tag{4.46a}$$

$$\dot{x}(t) = -c_1 \omega \sin(\omega t) + c_2 \omega \cos(\omega t) \tag{4.46b}$$

where $\omega = \sqrt{k}$, and the constant multiplicative terms may be solved with respect to the boundary conditions as $c_1 = x_0$ and $c_2 = v_0/\omega$.

The total system energy, which is the sum of the kinetic and potential energy terms, is defined as

$$E_{tot} = E_{potential} + E_{kinetic} \tag{4.47a}$$

$$E_{tot} = \frac{1}{2}kx^2 \quad + \frac{1}{2}\dot{x}^2 \tag{4.47b}$$

Because there are no dissipative terms in the acceleration, the total energy is a con-

served quantity, and is equal to its initial value ${}^0E_{tot}$ for all later times τ_k. Therefore, the constraint equation

$$g = {}^0E_{tot} - E_{tot} \equiv 0 \tag{4.48}$$

is exactly satisfied along the true system dynamics state space trajectory. During the process of numerical integration, Equation 4.48 will not be exactly satisfied, but will instead exhibit some small constraint violation

$$g(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k)) = {}^0E_{tot} - E_{tot}(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k)) \equiv \epsilon(\tau_k) \neq 0 \tag{4.49}$$

We seek a set of state correction factors $\boldsymbol{\Delta z}(\tau_k)$ that will serve to drive the instantaneous constraint violation to zero

$$g(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k) + \boldsymbol{\Delta z}(\tau_k)) \to 0 \tag{4.50}$$

Expanding the constraint equation $g(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k) + \boldsymbol{\Delta z}(\tau_k))$ yields

$$g(\boldsymbol{p}, \boldsymbol{z}^i(\tau_k) + \boldsymbol{\Delta z}(\tau_k)) = {}^0E_{tot} - \frac{1}{2}k\left(x(\tau_k) + \Delta x(\tau_k)\right)^2 - \frac{1}{2}\left(\dot{x}(\tau_k) + \Delta\dot{x}(\tau_k)\right)^2 \tag{4.51}$$

Recognizing the constraint violation term $\epsilon(\tau_k)$ from Equation 4.49 the substitution

$${}^0E_{tot} - \frac{1}{2}kx(\tau_k)^2 - \frac{1}{2}\dot{x}(\tau_k)^2 = \epsilon(\tau_k) \tag{4.52}$$

may be made. Neglecting terms of order $\boldsymbol{\Delta z}^2$, we find a bi-linear approximate expression for the constraint error

$$\epsilon(\tau_k) = kx(\tau_k)\Delta x(\tau_k) + \dot{x}(\tau_k)\Delta\dot{x}(\tau_k) \tag{4.53}$$

However, we are not free to independently vary $\Delta x(\tau_k)$ and $\Delta \dot{x}(\tau_k)$ as in the first order case, but instead are required to choose corrections such that the kinematic constraints of Equations 4.24 are obeyed. From Equations 4.25, we see the expression for the variation of position and velocity states as a function of the variation of the integrand coefficients ΔF. There are $N-1$ integrand coefficients F_0 through F_{N-2}, and $M+1$ sampled times τ_0 through τ_M. Using Equations 4.25, the constraint error for timestamp τ_0 is

$$\epsilon(\tau_0) = kx(\tau_0)\left[\Delta x(\tau_0)\right] + \dot{x}(\tau_0)\left[\Delta \dot{x}(\tau_0)\right] \tag{4.54a}$$

$$\epsilon(\tau_0) = kx(\tau_0)\left[\Lambda_{0,0}\Delta F_0 + \Lambda_{0,1}\Delta F_1 + ... + \Lambda_{0,N-2}\Delta F_{N-2}\right] + \tag{4.54b}$$
$$+ \dot{x}(\tau_0)\left[\Gamma_{0,0}\Delta F_0 + \Gamma_{0,1}\Delta F_1 + ... + \Gamma_{0,N-2}\Delta F_{N-2}\right]$$

$$\epsilon(\tau_0) = \left[kx(\tau_0)\Gamma_{0,0} + \dot{x}(\tau_0)\Lambda_{0,0}\right]\Delta F_0 + \left[kx(\tau_0)\Gamma_{0,1} + \dot{x}(\tau_0)\Lambda_{0,1}\right]\Delta F_1 + \tag{4.54c}$$
$$+ ... + \left[kx(\tau_0)\Gamma_{0,N-2} + \dot{x}(\tau_0)\Lambda_{0,N-2}\right]\Delta F_{N-2}$$

where $\Lambda_{i,j}$ and $\Gamma_{i,j}$ designate the element in the i^{th} row and j^{th} column of the matrix, using 0-based indexing. Similarly, expanding the constraint error at the k^{th} timestep τ_k reveals

$$\epsilon(\tau_k) = kx(\tau_k)\left[\Delta x(\tau_k)\right] + \dot{x}(\tau_k)\left[\Delta \dot{x}(\tau_k)\right] \tag{4.55a}$$

$$\epsilon(\tau_k) = kx(\tau_k)\left[\Lambda_{k,0}\Delta F_0 + \Lambda_{k,1}\Delta F_1 + ... + \Lambda_{k,N-2}\Delta F_{N-2}\right] + \tag{4.55b}$$
$$+ \dot{x}(\tau_k)\left[\Gamma_{k,0}\Delta F_0 + \Gamma_{k,1}\Delta F_1 + ... + \Gamma_{k,N-2}\Delta F_{N-2}\right]$$

$$\epsilon(\tau_k) = \left[kx(\tau_k)\Gamma_{k,0} + \dot{x}(\tau_k)\Lambda_{k,0}\right]\Delta F_0 + \left[kx(\tau_k)\Gamma_{k,1} + \dot{x}(\tau_k)\Lambda_{k,1}\right]\Delta F_1 + \tag{4.55c}$$
$$+ ... + \left[kx(\tau_k)\Gamma_{k,N-2} + \dot{x}(\tau_k)\Lambda_{k,N-2}\right]\Delta F_{N-2}$$

We may expand the constraint error at all $M+1$ CGL sampled timesteps as in

Equations 4.54 and 4.55, and stack the result into a vector/matrix equation of the form

$$\boldsymbol{\epsilon} = H\boldsymbol{\Delta F} \tag{4.56}$$

where H is a $(M+1)\times(N-1)$ matrix. By inspection of Equation 4.55c, the general form of matrix element $H_{i,j}$ can be seen to be

$$H_{i,j} = kx(\tau_i)\Gamma_{i,j} + \dot{x}(\tau_i)\Lambda_{i,j} \tag{4.57}$$

Equation 4.56 is a matrix expression of $M+1$ equations for $N-1$ unknowns, and since there are $M = N$ CGL sample points, this may be solved as a classical least squares problem [53]

$$\boldsymbol{\Delta F} = (H^T H)^{-1} H^T \boldsymbol{\epsilon} \tag{4.58}$$

Solving Equation 4.58 for the integrand coefficient correction $\boldsymbol{\Delta F}$, the constraint restoration state corrections may be calculated by substituting $F = F^{i-1} + \boldsymbol{\Delta F}$ into Equations 4.18 and 4.19.

The normal equations of least squares, as defined in Equation 4.58, are able to be used to solve for small constraint rectification terms within every MCPI iteration. However, slightly better results are obtained by using a weighted least squares formulation

$$\boldsymbol{\Delta F} = (H^T W H)^{-1} H^T W \boldsymbol{\epsilon} \tag{4.59}$$

where W is a diagonal weight matrix. Heuristic experience shows that the process of MCPI convergence does not occur evenly at every node during each MCPI iteration, but rather nearly sequentially by node, with the early nodes near the given initial boundary condition converging first, and then estimates of later (in time) nodes converging in later iterations. For this reason, the weighted least squares algorithm

of Equation 4.59 is preferable because it allows a larger weight to be placed on the earlier nodes which are more converged (and thus more closely aligned to the true dynamic solution at the intersection of constraint manifolds), and a lesser weight be assigned to the later nodes that are less converged. A very large weight is assigned to the first node at the initial time, indicating that the initial boundary condition is exact and should be treated as such. This approximately mimics solution of the least squares system with a constrained least squares algorithm. More rigorous constraint enforcement could be obtained by actually implementing a constrained least squares algorithm.

Choosing $k = 1$ (spring constant k in Equation 4.45), and $\boldsymbol{z}(t_0) = [1, 1]^T$, the benefit of this second order constrained MCPI algorithm is explored. We will first look at a short MCPI segment length of $t_f = \pi$ (the period of oscillation is 2π), with a modest MCPI order of approximation $N = 25$. The analytic solution for this short segment is shown in Figure 4.8. Figure 4.9 shows the magnitude of the difference between the MCPI numerical solution and the analytical solution for this short segment case, calculated at evenly sampled output timesteps. Both states are of order 1, and numerical precision of roughly 10^{-15} is achieved for both the position and velocity states, for both the second-order constrained MCPI algorithm and the uncorrected second-order MCPI algorithm. It is clear that both algorithms are able to achieve numerical precision solutions accurate to the limit of double precision algebra, even when reporting the solution at evenly sampled timesteps that lie in between the CGL cosine-sampled timesteps. Figure 4.10 shows the *a posteriori* calculated constraint violation (as defined in Equation 4.52), calculated at the same evenly sampled output timesteps. The errors for both algorithms are within numerical precision, and are of the same order as the errors from the analytic solution. Therefore, the constrained algorithm and the uncorrected algorithm are

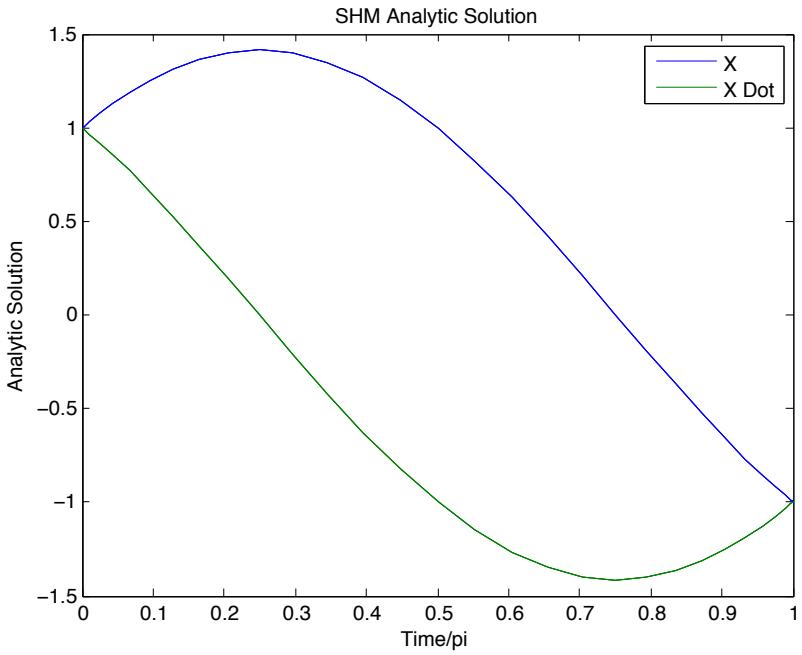


Figure 4.8: Simple harmonic motion, analytic solution for short segment with $t_f = \pi$.

evenly matched when it comes to solution accuracy. However, the advantage of the constrained MCPI method becomes clear when examining Figure 4.11, which shows the convergence trends of the algorithms. The uncorrected algorithm converges in 14 iterations, whereas the constrained algorithm converges in only 4 iterations. The geometric rate of convergence is, remarkably, over 3 orders of magnitude per Picard iteration. Note that the initial estimate of the position and velocity states was completely uninformed, the initial boundary condition was used at every CGL node. The constrained MCPI method is able to converge in 3.5X fewer iterations for this short segment case.

Choosing now a segment length of $t_f = 5\pi$, the benefits of the constrained method with be explored for a longer MCPI segment of approximation, with a higher order of approximation ($N = 90$). The analytic solution is shown in Figure 4.12. An inter-

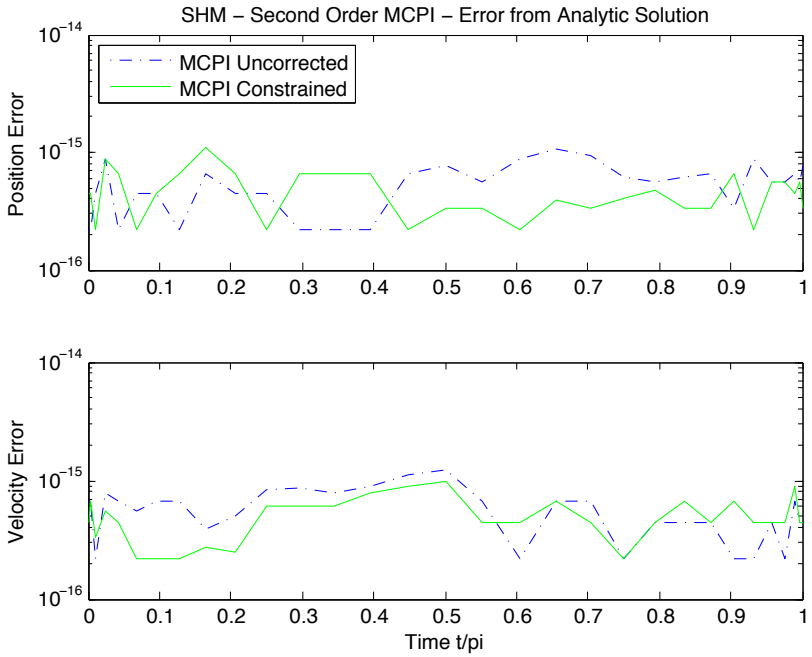


Figure 4.9: Simple harmonic motion, MCPI solution error from analytic solution for short segment with $t_f = \pi$, constrained and uncorrected MCPI.

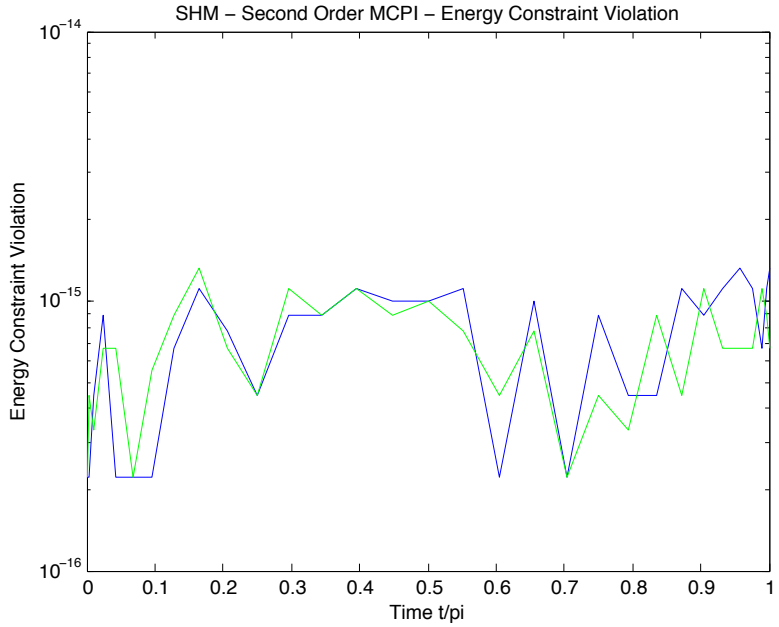


Figure 4.10: Simple harmonic motion, MCPI constraint error for short segment with $t_f = \pi$, constrained and uncorrected MCPI.

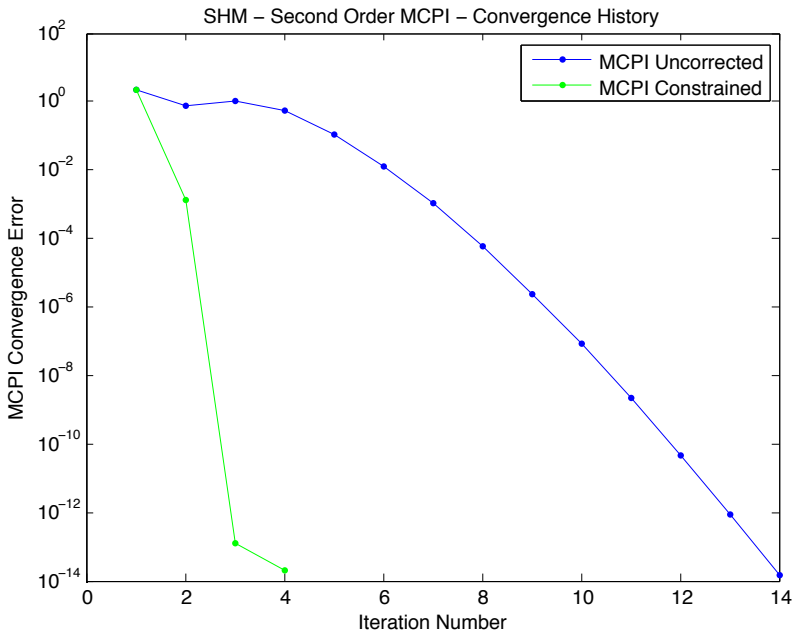


Figure 4.11: Simple harmonic motion, MCPI convergence trends for short segment with $t_f = \pi$, constrained and uncorrected MCPI.

esting phenomenon is seen by looking at the convergence trends of the constrained MCPI algorithm and the uncorrected MCPI algorithm, as shown in Figure 4.13. The unconstrained method is not able to converge within the maximum number of allowed iterations, indicative of the fact that the Chebyshev order of approximation is too low for the chosen segment length. Note that this is a less catastrophic failure mode than in the first-order MCPI results of the previous section, where the method would completely diverge. This is likely due to the kinematic constraint enforcement between the position and velocity states, which serves to keep the error from blowing up to infinity. The constrained method is able to converge to within the (fairly lenient) convergence threshold, and is able to do so in far fewer iterations than it took the unconstrained method to achieve its noise floor values. There is a slight oscillation of the convergence error of the constrained method near the end, again due to

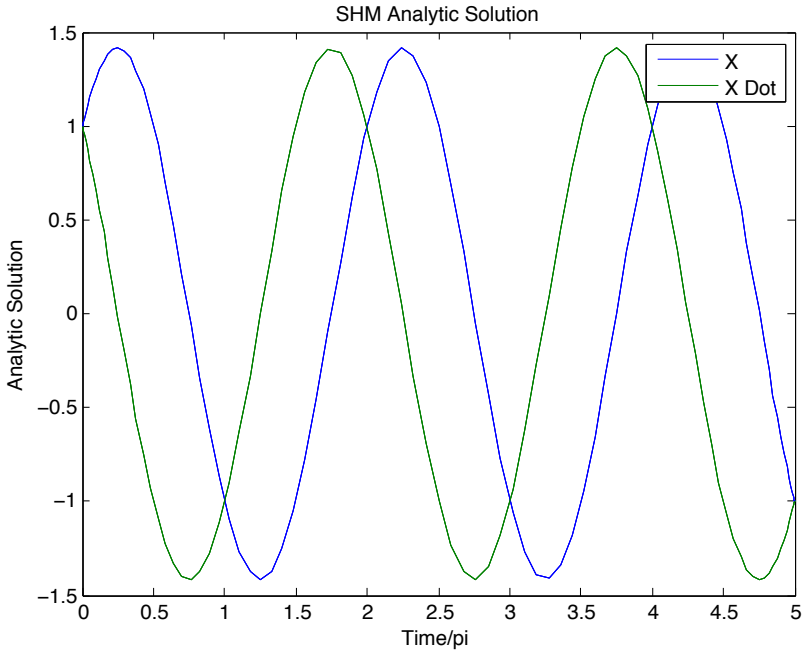


Figure 4.12: Simple harmonic motion, analytic solution for long segment with $t_f = 5\pi$.

the fact that segment length upper limit is being pushed for the chosen Chebyshev order of approximation. During the initial iterations when the approximation error is large, it is necessary to perform the constraint correction step more than once in order to achieve the convergence trend that is shown in the figure. This is due to the fact that the correction only keeps linear terms in the expansion with respect to the correction factors, and when the corrections are large the linear terms are not a representative approximation of the truth.

Figure 4.14 shows the MCPI approximation error with respect to the analytic solution. The constraint error is shown in Figure 4.15, again calculated at evenly sampled timesteps. Neither algorithm is able to achieve numeric precision due to inadequate Chebyshev order of approximation, however the constrained method is 2-3 orders of magnitude closer to the analytic solution, and to conserving the energy.

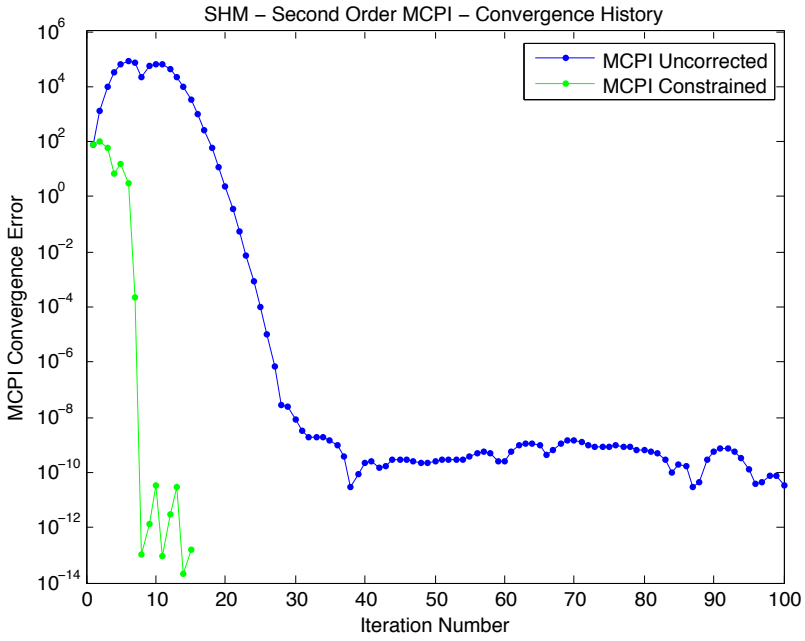


Figure 4.13: Simple harmonic motion, MCPI convergence trends for long segment with $t_f = 5\pi$, constrained and uncorrected MCPI.

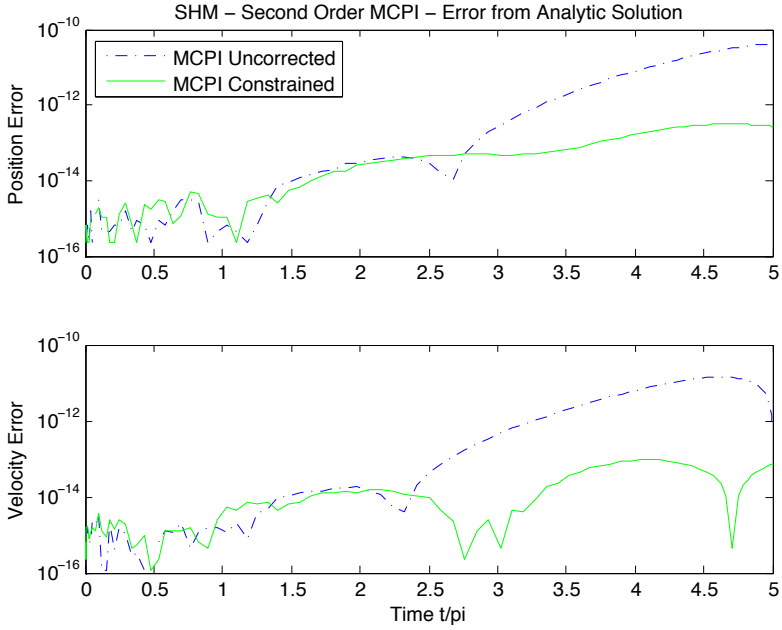


Figure 4.14: Simple harmonic motion, MCPI solution error from analytic solution for long segment with $t_f = 5\pi$, constrained and uncorrected MCPI.

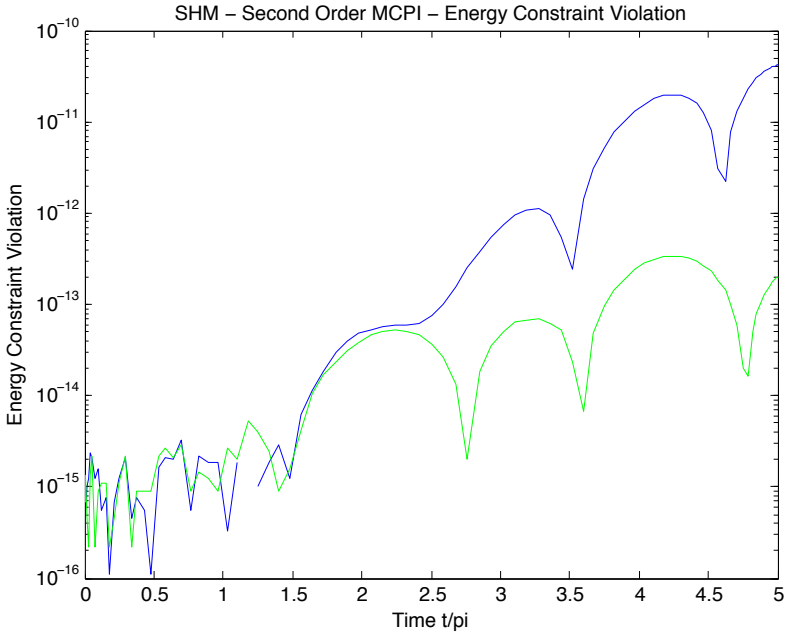


Figure 4.15: Simple harmonic motion, MCPI constraint error for long segment with $t_f = 5\pi$, constrained and uncorrected MCPI.

Summarizing the above example, two cases were presented to demonstrate second-order constrained MCPI on a simple harmonic oscillator scalar ODE system. A short segment of approximation, with a low Chebyshev order of approximation demonstrated that the constrained, as well as the uncorrected MCPI second-order methods are capable of achieving arbitrary accuracy within the limits of double precision. However, the constrained method does so in 3.5 times fewer iterations. A longer segment of approximation, with a Chebyshev order of approximation intentionally chosen to be slightly inadequate for the segment length, demonstrated that the constrained method is able to converge in a case where the unconstrained method is not (within the maximum allowed number of iterations). The constrained method converges in roughly 15 iterations, whereas the unconstrained method delivers 2-3 orders of magnitude worse fidelity solution, taking 100 iterations to do so.

5. SERIAL AND PARALLEL MCPI IMPLEMENTATIONS

5.1 MCPI Serial Libraries

MCPI function libraries have been created to encourage widespread use of the MCPI method for solution of Ordinary Differential Equations. These libraries are a set of efficient and lightweight classes for solution of Initial Value Problems (IVPs). The goal of the project is to create an easy to use toolset that effectively eliminates the learning curve of using MCPI methods, but at the same time is versatile and powerful enough for application to a variety of projects. The user is not required to have a thorough understanding of the inner-workings of MCPI in order to implement it in their own projects. Essentially, if the user is familiar with Matlab's `ODE45`, they will be able to operate the MCPI integrator. Solvable ODEs can be linear or non-linear, autonomous or non-autonomous, and first-order or second-order. Higher order systems are solvable by decomposition to a first-order or second-order system through the inclusion of additional states that are the time derivatives of lower order states. The MCPI library code was first presented in 2013 [67], but has undergone several redesigns since first publication.

Figure 5.1 shows a high-level overview of the function library structure from an implementation point of view. The user provides a pointer to an integrand function for the problem at hand, that is, the update function that describes how the time derivatives of the system states behave. Additionally, the user provides the relevant boundary conditions for the system states, defined at the initial time. If the system has time-varying parameters, or other numerical data is required in the integrand function, these may be inputted as well. Given these inputs, the MCPI library functions will numerically solve the state-space trajectories of the system over the

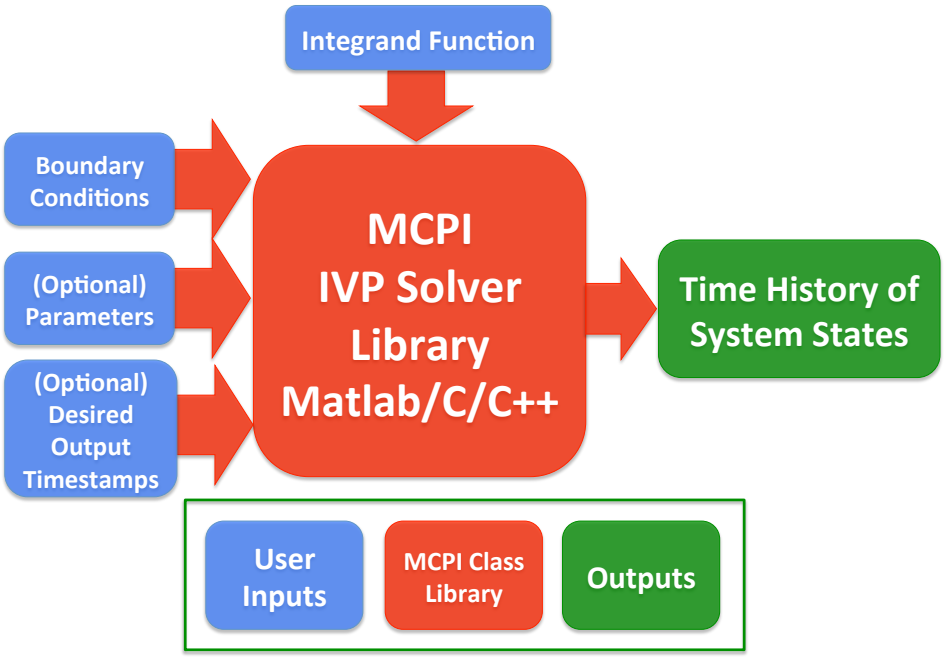


Figure 5.1: Flowchart of MCPI libraries.

desired time interval. The desired output times may be specified by the user, in which case the MCPI library code will interpolate and return the solution at the user's desired times. If no output times are specified, the solution will be returned at the CGL nodes.

The MCPI libraries are available in Matlab, C, and C++. They are fully cross-platform, and has been tested on Windows, Linux, and Apple computers. The structure of the Matlab functions is hierarchical, with an abstract parent class and derived child classes tailored to the solution of various problem types. This modular approach is to allow for future expansion, or application-specific customization and optimization. The C library uses "1-based" array indexing, and column-major array storage in order to be compatible with Fortran, which many legacy aerospace code-

bases still utilize. The C++ version consists of wrapper functions around the core C code.

5.2 Parallel Propagator Framework

A framework for parallel numerical propagation of large numbers of perturbed orbits with MCPI has been developed [52]. This framework consists of three modules; a main (control) module, a set of MCPI worker modules, and a renderer module, as illustrated schematically in Figure 5.2. The control module interacts with the satellite catalog and sends jobs to the MCPI worker processes. We use an SQLite database to manage the space object catalog, and have created functions to import Two-Line Element (TLE) data to initialize or update the catalog. Using SQLite syntax, a user is able to query and propagate the full catalog or generalized subsets of orbits. The MCPI worker processes are fed data from the main process, propagate an orbital trajectory with user-specified perturbation force models, and report the propagated ephemeris to the renderer module and back to the main module for catalog update. The MCPI worker modules have automated internal tuning parameter selection (using the tuning parameter selection look-up table as described in Section 3) such that the user is required to specify only a desired physical accuracy tolerance. Currently the tuning parameters are selected prior to propagating an MCPI segment, but a later version of this software package will have in-the-loop adaptive tuning parameter adjustment, starting from the initial values in the *a priori* tuning schemes currently in use. The renderer module displays the propagated satellite trajectories around the Earth in an OpenGL environment while the worker modules calculate further ephemerides points in parallel. It has basic camera controls to allow the user to rotate the field of view and zoom on an object or area of interest. All three modules have been developed using standard C/C++ libraries,

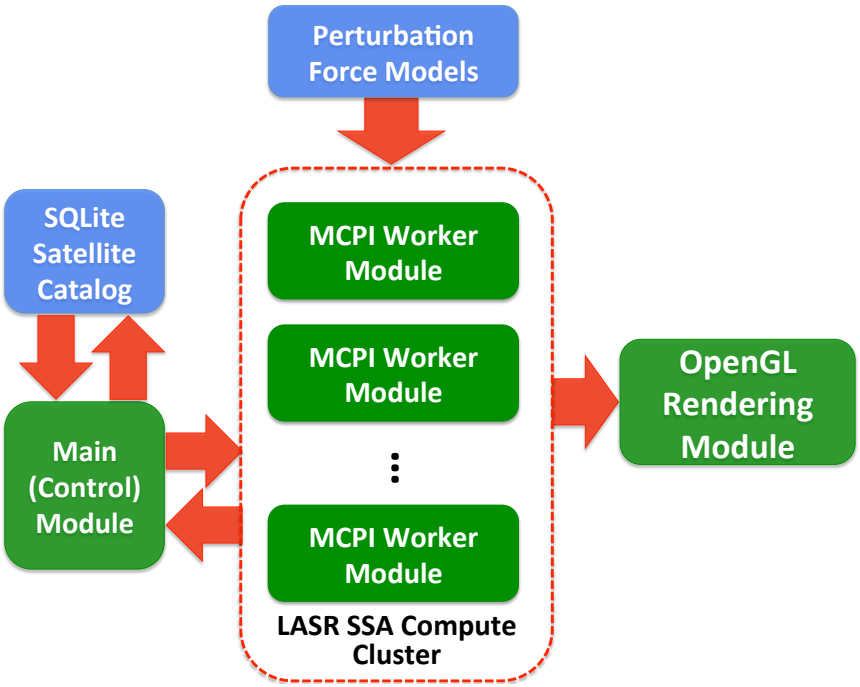


Figure 5.2: Flowchart of MCPI parallel propagator framework.

and the Message Passing Interface (MPI) for inter-process communication.

The control module and MCPI worker modules are capable of running on virtually any computer with a multi-core CPU, and were developed on a linux desktop computer with an Intel i7 processor. The renderer module utilizes GPU acceleration, but should run on any computer with an OpenGL-capable graphics card. This framework is fully scalable and was designed for the compute cluster environment. The Land, Air, and Space Robotics (LASR) Laboratory at Texas A&M University has a cluster computer called the LASR SSA Cluster. It consists of 16 nodes, each with two Intel Xeon 2.6GHz 6-(virtual)-core CPUs and 64GB of RAM. The cluster has a combined 192 virtual cores (with threading), which means 192 parallel MCPI worker processes can be run. More detailed specifications of the LASR SSA Cluster are given in Appendix A.

Two example applications of the parallel MCPI framework are presented: a parallel space catalog propagator, and a parallel uncertainty propagation suite. The code is fully extensible, and was designed with future applications in mind. It is modularized to allow for wrapping the entire package within a catalog maintenance or conjunction analysis toolset, and it was purposely designed for easy addition of user-supplied perturbation force models. This framework will be utilized in a forthcoming MCPI paper on benchmarking several state-of-practice numerical propagators using a realistic set of test cases, as well as serving as a baseline with which to compare forthcoming MCPI massively parallel implementations.

5.2.1 Example: Space Catalog Propagation

The MCPI parallel propagator framework is used to numerically propagate a similar catalog to the publicly available Celestrak Two-Line Element (TLE) orbital catalog, which contained about 15000 objects at the time of download. Images for public release are generated by perturbing the orbital elements of the Celestrak catalog to generate a new catalog with the same population statistics, but different osculating orbital elements. This psuedo-catalog is then propagated forward in time using the parallel propagator framework, as shown in the screen capture from the parallel propagator renderer module in Figure 5.3. The satellite and debris objects are color coded in the renderer according to the logic in Table 5.1. The renderer module displays the calculated ephmerides at roughly 3600x real-time (1s of display $\simeq$ 1hr of simulated orbit time). Using 192 MCPI worker processes, the LASR SSA cluster is able to propagate the large pseudo-Celestrak database in roughly equal propagation time as the speeded-up display time, so the renderer displays a batch of orbits in roughly the same amount of time that it takes to calculate the next batch.

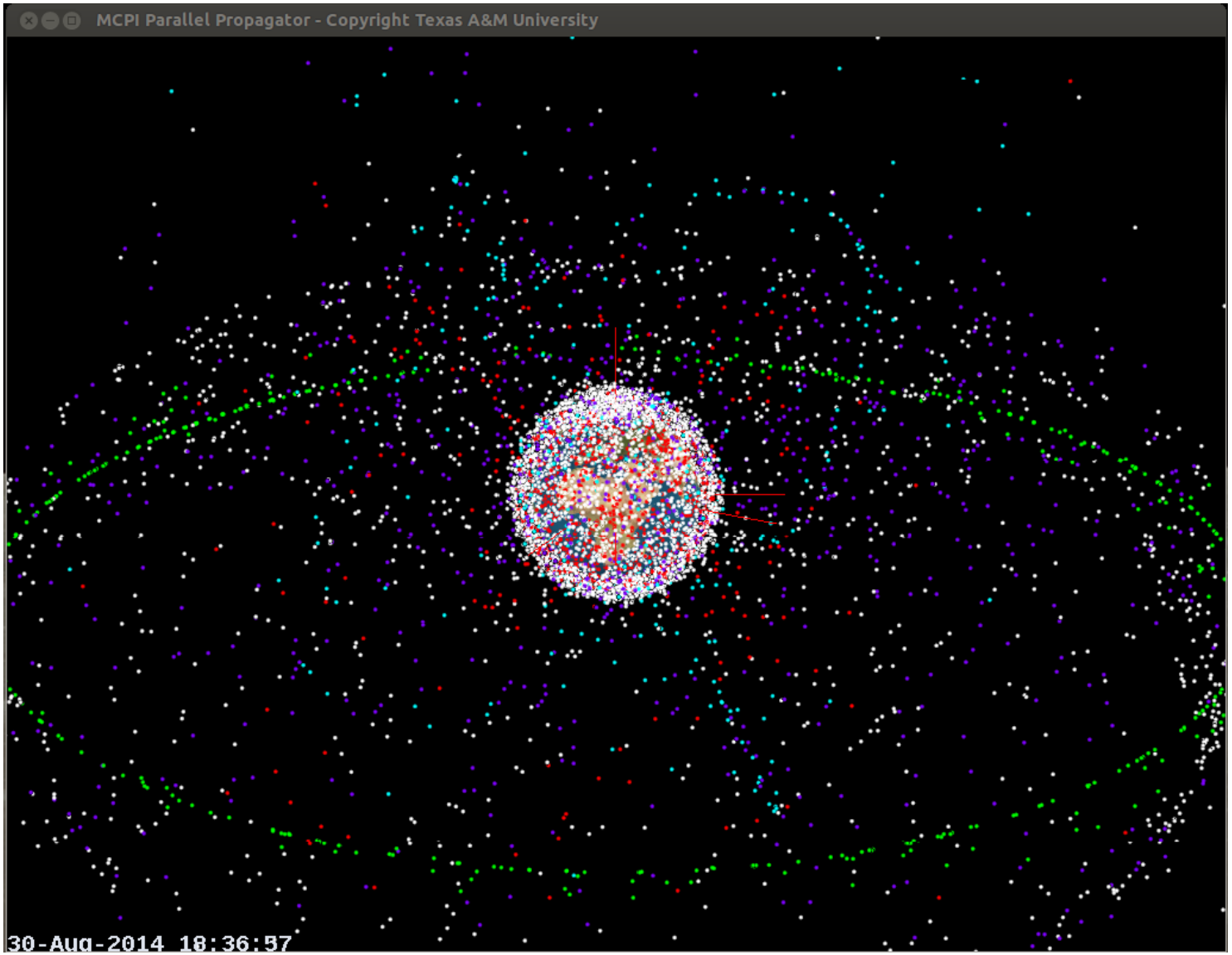


Figure 5.3: Screenshot of MCPI parallel propagator renderer, propagating 15000 object space catalog.

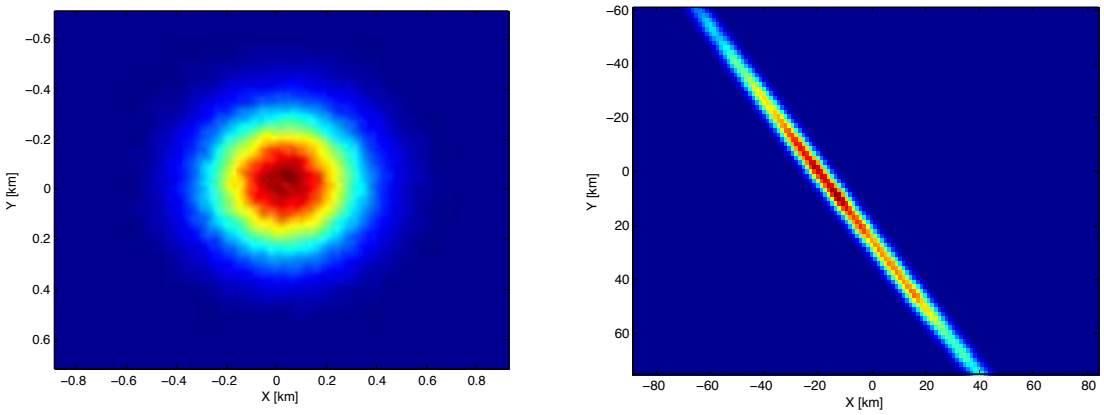
Table 5.1: Parallel propagator renderer output color scheme

Color	Description	SQLite Selection Criteria
Red	Debris From Major Breakup Events	Name contains 'iridium33', 'fengyun 1', 'cosmos 2251', or 'breeze-m'
Green	GEO Belt	$a \simeq 42,164$km and $i \simeq 0°$
Turquoise	Molniya and Critically Inclined	$i \simeq 63.4°$
Purple	Rocket Bodies	Name contains 'r/b'
White	Other	All that do not fit above criteria

5.2.2 Example: Uncertainty Propagation

The MCPI parallel propagator framework is used to propagate a batch of 20000 particles in a Monte Carlo-style dispersion analysis. The particles are described by their initial distribution about an ISS-like LEO reference orbit. Initially the particles are are distributed in a three-dimensional Gaussian density, with $(\sigma_x, \sigma_y, \sigma_z)$ standard deviation of 0.2km in Earth-Centered-Inertial coordinates. An (x, y) cross-section of the initial probability density is shown in Figure 5.4a. Forward propagating for 1.5 days (about 24 orbits) with 40th-order spherical harmonic gravity perturbations yields the final probability density function shown in Figure 5.4b.

This example problem is not intended to be groundbreaking science, but rather to demonstrate the time-saving potential of the parallel propagation framework. The code utilizes several of the efficiency improvements described in Section 2, namely: *(i)* radially adaptive gravity, *(ii)* zeroth-order Taylor series gravity local approximations, *(iii)* F&G analytic solution warm/hot start. Running in serial on a single desktop workstation, the perturbed orbit propagation of the 20000 particles requires roughly three and a quarter hours of computation time. With 192 parallel MCPI worker processes on the cluster, it completes in about one minute. These quoted computation times do not yet leverage the extreme computational savings benefit of the first-order Taylor series approximate gravity model for Monte Carlo-type problems as described in Section 2.3. Once this improvement is built into the parallel



(a) Initial (x, y) probability density.

(b) Final (x, y) probability density.

Figure 5.4: Monte Carlo propagation of 20000 particle system on ISS-like orbit over 1.5 days (24 orbits).

propagator framework, the time to propagate the particle cloud in the vicinity of the reference trajectory will be reduced by orders of magnitude.

6. CONCLUSION

6.1 Overview

This dissertation has presented a body of work designed to make Modified Chebyshev Picard Iteration more efficient and automated when applied to the problem of perturbed orbit propagation. Additionally, a methodology was presented that greatly increases MCPI performance on Ordinary Differential Equation systems for which there are intrinsically conserved quantities. Efficient implementations of MCPI are presented that allow a user with limited familiarity of the inner-workings of MCPI to easily and efficiently numerically integrate ODE systems. A detailed term-by-term derivation was presented for first order MCPI, as well as for a novel second order MCPI method that rigorously enforces the kinematic constraint in the Chebyshev coefficients between the acceleration, velocity, and position level states.

Using pre-existing methods, numerically integrating the equations of perturbed orbital motion is computationally very expensive. By far the most costly part is the evaluation of the high-fidelity perturbation force functions, spherical harmonic gravity in particular. Because MCPI is an iterative method, and the perturbation accelerations must be evaluated every iteration, there are two strategies for reducing the total computation cost: *(i)* reducing the number of required iterations, and *(ii)* reducing the computational cost per iteration. Strategies for doing both have been presented for MCPI. These strategies are generally applicable to broad classes of numerical integration algorithms, and result in orders of magnitude performance improvement.

Until now, MCPI has required a user-in-the-loop to choose appropriate values of MCPI tuning parameters for each particular orbit and segment propagated. This

is not conducive to practical applications of MCPI, for instance propagating large space catalogs quickly. Additionally it invites poor implementations of MCPI, and in the literature some investigators have reached invalid assessments due to sub-optimal tuning parameter selection. Two methods for generating tuning parameter sets for MCPI were presented. A tuning parameter lookup table has been created using the results of these studies, the result of which is a fully autonomous algorithm capable of propagating large batches of orbits at user-specified fidelity, without user tuning intervention.

A framework for "constraint restoration" has been presented that greatly improves the performance of MCPI when applied to systems with intrinsic constants of motion. A first order and a second order MCPI constraint restoration algorithm has been demonstrated to greatly reduce the required number of MCPI iterations, increase the segment length of convergence, and provide orders of magnitude better solution accuracy for the same convergence threshold. This new method shows promise on a small number of example problems and should be further studied, especially the application of the method to orbit propagation.

Optimized MCPI implementations have been presented that leverage most of the developments described above, and allow a user with limited familiarity of MCPI to quickly use the algorithm for their applications. A serial implementation is applicable to general ODE problems, and has been implemented in Matlab as well as in compiled languages. A parallel wrapper around the serial version has been created to allow MCPI to propagate large batches of orbits in parallel, in a cluster environment. Example applications are presented that demonstrate the utility of the MCPI implementations and the algorithmic improvements described above.

6.2 Recommended Future Work

As a result of the research efforts undertaken during this dissertation, several promising topics for future research have become clear:

Locally Adaptive MCPI Algorithm: The *a priori* tuning parameter sets described in Sections 3.3 and 3.4 allow MCPI to autonomously propagate perturbed orbital trajectories to user-specified precision. However, they were designed for only the equations of orbit propagation, and considering only spherical harmonic gravity perturbations. The locally adaptive tuning method that is briefly described in Section 3.6 applies to general ODE systems, including application to perturbed orbit propagation with any conceivable perturbing acceleration. Based upon comparison of the empirical tuning set with the genetic algorithm set, it seems that the empirical tuning set is only slightly sub-optimal, and the development of the locally adaptive method is the most productive use of further research effort.

Application of Constraint Restoration MCPI to Orbit Propagation: The Minimum Correction Constraint Restoration framework presented in Section 4 has been shown to greatly reduce the required number of MCPI iterations, as well as increase the feasible segment length of convergence. However the demonstrations to date have been on simple 2^{nd} or 3^{rd} order dynamical systems with one or more exact integrals. Application of this method to orbit propagation will likely have a large positive impact on the performance of MCPI. Conservative perturbation forces will conserve the Hamiltonian, and non-conservative forces will conserve the total work/energy integral. The Initial Value Problem MCPI formulation is already able to converge for segment lengths of several orbits, however the Boundary Value Problem MCPI formulation is only able to

converge over a fraction of one orbit. Application of the constraint restoration could help expand the feasible segment length of convergence of the Boundary Value Problem formulation. Additionally, the six degree-of-freedom coupled orbit/attitude problem stands to gain from the constraint restoration MCPI. The six DoF problem is known to be a "stiff" ODE, meaning it exhibits dynamics at short and long timescales (short period from rigid body tumbling, long period from orbital motion). However, attitude propagation has several inherently conserved quantities (rigid body angular momentum/energy for the case of torque free motion, quaternion unit norm constraint). Propagating the six DoF system is limited by the feasible segment length dictated by the short period motion and coupling effects. Using the constraint restoration method to expand the segment length may have a large impact on the efficiency of coupled attitude/orbit propagation with MCPI. This is an important area for further study.

Extension of Parallel Propagator Framework for SSA Applications: The parallel propagator framework described in Section 5.2 was designed with future applications in mind. The current example applications of parallel catalog propagation and uncertainty analysis are just the tip of the iceberg. Demonstrations of parallel catalog maintenance (batch least squares orbit determination) or parallel conjunction analysis with MCPI are natural extensions of the existing work, and would make powerful demonstrations of the utility of MCPI to propagate orbits in parallel. Many SSA problems (data association, conjunction analysis, probability of collision, and Monte Carlo studies) require many orbits to be quickly propagated. MCPI and related algorithms therefore have a wide set of potential applications.

Massively Parallel MCPI: MCPI is an inherently parallelizable algorithm. Bai explored the possibilities of GPU parallelization, and showed that the computational speedups can be large. A massively parallel MCPI framework needs to be developed, built to the same coding standards and modularity as the serial MCPI implementations described in Section 5. Applications such as real-time embedded optimal control (model predictive optimal control) for spacecraft maneuvers or UAV path planning would make excellent demonstrations of the method's power. Parallel evaluations of the perturbation force functions will revolutionize the way orbit propagation is done, and therefore open the door to solve many SSA challenges.

REFERENCES

[1] P. D. Nielsen, K. T. Alfriend, M. J. Bloomfield, and J. T. Emmert et al., *Continuing Kepler's Quest: Assesing Air Force Space Command's Astrodynamics Standards*, The National Academies Press, 2012.

[2] D. J. Kessler and B. G. Cour-Palais, "Collision Frequency of Artificial Satellites: The Creation of a Debris Belt," *Journal of Geophysical Research*, vol. 83, pp. 2637–2646, 1978.

[3] A. Iserles, *A First Course in Numerical Analysis of Differential Equations*, Cambridge University Press, second edition, 2009.

[4] J. C. Butcher, *Numerical Methods for Ordinary Differential Equations*, John Wiley & Sons, 2008.

[5] J. C. Butcher, "Implicit Runge-Kutta Processes," *Mathematics of Computation*, vol. 18, pp. 50–64, 1964.

[6] J. R. Dormand and P. J. Prince, "A Family of Embedded Runge-Kutta Formulae," *Journal of Computational and Applied Mathematics*, vol. 6, pp. 19–26, 1980.

[7] J. R. Dormand and P. J. Prince., "A Reconsideration of Some Embedded Runge-Kutta Formulae," *Journal of Computational and Applied Mathematics*, vol. 15, pp. 203–211, 1986.

[8] H. A. Watts, "Starting Step Size for an ODE Solver," *Journal of Computational and Applied Mathematics*, vol. 9, pp. 177–191, 1983.

[9] L. Gladwell, L. F. Shampine, and R. W. Brankin, "Automatic Selection of the Initial Step Size for an ODE Solver," *Journal of Computational and Applied Mathematics*, vol. 18, pp. 175–192, 1987.

[10] O. Montenbrook and E. Gill, *Satellite Orbits*, Springer, 2000.

[11] J. R. Dormand, M. E. A. El-Mikkawy, and P. J. Prince, "High-Order Embedded Runge-Kutta-Nystrom Formulae," *IMA Journal of Numerical Analysis*, vol. 7, pp. 423–430, 1987.

[12] G. Wanner and E. Hairer, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, Springer-Verlag, 1991.

[13] K. Wright, "Some Relationships Between Implicit Runge-Kutta, Collocation, and Lanczos τ Methods, and Their Stability Properties," *BIT Numerical Mathematics*, vol. 10, pp. 217–227, 1970.

[14] B. A. Jones and R. L. Anderson, "A Survey of Symplectic and Collocation Methods for Orbit Propagation," in *22nd Annual AAS/AIAA Space Flight Mechanics Meeting, Charleston, SC*, 2012.

[15] B. K. Bradley, B. A. Jones, G. Beylkin, K. Sandberg, and P. Axelrad, "Bandlimited Implicit Runge Kutta Integration for Astrodynamics," *Celestial Mechanics and Dynamical Astronomy*, vol. 119, pp. 143–168, 2014.

[16] B. A. Jones, "Orbit Propagation Using Gauss-Legendre Collocation," in *AIAA/AAS Astrodynamics Specialist Conference*, Minneapolis, MN, 2012.

[17] J. M. Aristoff and A. B. Poore, "Implicit Runge Kutta Methods for Orbit Propagation," in *AIAA/AAS Astrodynamics Specialist Conference*, Minneapolis, MN, 2012.

[18] J. M. Aristoff, J. T. Horwood, and A. B. Poore, "Orbit and Uncertainty Propagation: a Comparison of Gauss-Legendre, Dormand-Prince, and Chebyshev-Picard-Based Approaches," *Celestial Mechanics and Dynamical Astronomy*, vol. 118, pp. 13–28, 2013.

[19] J.M. Aristoff, J. T. Horwood, and A. B. Poore, "Implicit Runge Kutta Methods for Uncertainty Propagation," in *Advanced Maui Optical and Space Surveillance Technologies Conference,*, Maui, HI, 2012.

[20] B. K. Bradley, B. A. Jones, G. Beylkin, and P. Axelrad, "A New Numerical Integration Technique in Astrodynamics," in *22nd Annual AAS/AIAA Space Flight Mechanics Meeting, Charleston, SC*, 2012.

[21] M. M. Berry and L. M. Healy, "Implementation of Gauss-Jackson Integration for Orbit Propagation," *The Journal of Astronautical Sciences*, vol. 52, pp. 331–357, 2004.

[22] J. L. Maury Jr. and G. P. Seagal, "Cowell Type Numerical Integration as Applied to Satellite Orbit Computation," Tech. Rep., Goddard Space Flight Center, Greenbelt, MD, 1969.

[23] M. M. Berry, "A Variable-Step Double-Integration Multi-Step Integrator," Ph.D. dissertation, Virginia Polytechnic Institute and State University, 2004.

[24] K. Fox, "Numerical Integration of the Equations of Motion of Celestial Mechanics," *Celestial Mechanics*, vol. 33, pp. 127–142, 1984.

[25] O. Montenbrook, "Numerical Integration Methods for Orbital Motion," *Celestial Mechanics and Dynamical Astronomy*, vol. 53, pp. 59–69, 1992.

[26] X. Bai and J. L. Junkins, "Modified Chebyshev-Picard Iteration Methods for Solution of Initial Value Problems," *Advances in the Astronautical Sciences*,

vol. 139, pp. 345–362, 2011.

[27] C. W. Clenshaw and H. J. Norton, "The Solution of Nonlinear Ordinary Differential Equations in Chebyshev Series," *The Computer Journal*, vol. 6, no. 1, pp. 88–92, 1963.

[28] J. S. Shaver, "Formulation and Evaluation of Parallel Algorithms for the Orbit Determination Problem," Ph.D. dissertation, Massachusetts Institute of Technology, March 1980.

[29] T. Feagin and P. Nacozy, "Matrix Formulation of the Picard Method for Parallel Computation," *Celestial Mechanics and Dynamical Astronomy*, vol. 29, pp. 107–115, 1983.

[30] T. Fukushima, "Vector Integration of Dynamical Motions by the Picard-Chebyshev Method," *The Astronomical Journal*, vol. 113, pp. 2325 2328, 1997.

[31] X. Bai, "Modified Chebyshev-Picard Iteration Methods for Solution of Initial Value and Boundary Value Problems," Ph.D. dissertation, Texas A&M University, 2012.

[32] X. Bai and J. L. Junkins, "Modified Chebyshev-Picard Iteration Methods for Orbit Propagation," *Journal of the Astronautical Sciences*, vol. 58, pp. 583–613, 2011.

[33] X. Bai and J. L. Junkins, "Modified Chebyshev-Picard Iteration Methods for Solution of Boundary Value Problems," *Advances in the Astronautical Sciences*, vol. 140, pp. 381–400, 2011.

[34] X. Bai and J. L. Junkins, "Modified Chebyshev Picard Iteration Methods for Station-Keeping of Translunar Halo Orbits," *Mathematical Problems in Engineering*, vol. 2012, pp. 1–18, 2012.

[35] A. Bani-Younes, "Orthogal Polynomial Approximation in Higher Dimensions: Applications in Astrodynamics," Ph.D. dissertation, Texas A&M University, 2013.

[36] E. Picard, "Sur l'Application des Méthodes d'Approximations Successives à l'étude de Certaines équations Différentielles Ordinaires," *Journal de Mathématiques Pures et Appliquées*, vol. 9, pp. 217–272, 1893.

[37] R. Woollands, A. Bani Younes, and J. L. Junkins, "A New Solution for the General Lambert Problem," in *37th Annual AAS Guidance & Control Conference*, Breckenridge, CO, 2014.

[38] R. M. Woollands, J. L. Read, B. Macomber, A. Probe, A. Bani Younes, and J. L. Junkins, "Method of Particular Solutions and Kustaanheimo-Stiefel Regularized Picard Iteration for Solving Two-Point Boundary Value Problems," in *25th AIAA/AAS Spaceflight Mechanics Meeting*, Williamsburg, VA, 2015.

[39] J. L. Junkins, A. Bani-Younes, R. Woollands, and X. Bai, "Orthogonal Polynomial Approximation in Higher Dimensions: Applications in Astrodynamics," in *Jer-Nan Juang Astrodynamics Symposium, College Station, TX*, 2012.

[40] L. Fox and I. B. Parker, *Chebyshev Polynomials in Numerical Analysis*, Oxford University Press, 1968.

[41] H. Schaub and J. L. Junkins, *Analytical Mechanics of Space Systems*, American Institute of Aeronautics and Astronautics, third edition, 2014.

[42] D. A. Vallado, *Fundamentals of Astrodynamics and Applications*, Microcosm Press, fourth edition, 2013.

[43] J. L. Junkins, "Investigation of Finite-Element Representations of the Geopotential," *AIAA Journal*, vol. 14, pp. 803–808, 1976.

[44] P. Singla and J. L. Junkins, *Multi-Resolution Methods for Modeling and Control of Dynamical Systems*, CRC Press, 2009.

[45] R. C. Engels and J. L. Junkins, "Local Representation of the Geopotential by Weighted Orthonomal Polynomials," *Journal of Guidance, Control, and Dynamics*, vol. 3, pp. 55–61, 1980.

[46] D. A. Vallado, "An Analysis of State Vector Propagation Using Differing Flight Dynamics Programs," in *AAS/AIAA Space Flight Mechanics Conference*, Copper Mountain, CO, 2005.

[47] N.K. Pavlis, S.A. Holmes, S.C. Kenyon, and J.K. Factor, "An Earth Gravitational Model to Degree 2160: EGM2008," in *General Assembly of the European Geosciences Union, Vienna, Austria*, 2008.

[48] A. Probe, B. Macomber, R. Woollands, and J. L. Junkins, "Radially Adaptive Evaluation of the Spherical Harmonic Gravity Series for Numerical Orbit Propagation," in *25th AAS/AIAA Space Flight Mechanics Meeting*, Williamsburg, VA, 2015.

[49] P. Leopardi, "A Partition of the Unit Sphere Into Regions of Equal Area and Small Diameter," *Electronic Transactions on Numerical Analysis*, vol. 25, pp. 309–327, 2006.

[50] A. Probe, B. Macomber, D. Kim, R. Woollands, and J. L. Junkins, "Terminal Convergence Approximation Modified Chebyshev Picard Iteration for Efficient Numerical Integration of Orbital Trajectories," in *Advanced Maui Optical and Space Surveillance Technologies Conference*, Maui, Hawaii, 2014.

[51] J. Read, J. L. Junkins, and A. Bani-Younes, "State Transition Matrix Propagation for Perturbed Orbital Motion Using Modified Chebyshev Picard Iteration,"

in *38th Annual AAS Guidance & Control Conference*, Breckenridge, CO, 2015.

[52] B. Macomber, A. Probe, R. Woollands, and J. L. Junkins, "Modified Chebyshev Picard Iteration for Orbit Catalog Propagation and Monte Carlo Analysis," in *38th Annual AAS Guidance & Control Conference*, Breckenridge, CO, 2015.

[53] J. L. Crassidis and J. L. Junkins, *Optimal Estimation of Dynamic Systems*, Chapman & Hall/CRC, first edition, 2004.

[54] Y. Kozai, "The Motion of a Close Earth Satellite," *The Astronomical Journal*, vol. 64, pp. 367–377, 1959.

[55] D. Brouwer, "Solution of the Problem of Artificial Satellite Theory Without Drag," *The Astronomical Journal*, vol. 64, pp. 378–396, 1959.

[56] D. Brouwer and G.I. Hori, "Theoretical Evaluation of Atmospheric Drag Effects in the Motion of an Artificial Satellite," *The Astronomical Journal*, vol. 66, pp. 193–225, 1961.

[57] R. H. Lyddane, "Small Eccentricities or Inclinations in the Brouwer Theory of the Artificial Satellite," *The Astronomical Journal*, vol. 68, pp. 555–558, 1963.

[58] F. R. Hoots and R. L. Roehrich, "Spacetrack Report No. 3: Models for Propagation of NORAD Element Sets," Tech. Rep., 1980.

[59] F. R. Hoots, P. W. Schumacher Jr., and R. A. Glover, "History of Analytical Orbit Modeling in the U.S. Space Surveillance System," *Journal of Guidance, Control, and Dynamics*, vol. 27, pp. 174–185, 2004.

[60] D. A. Vallado, P. Crawford, and R. Hujsak, "Revisiting Spacetrack Report #3," in *AIAA/AAS Astrodynamics Specialist Conference*, Keystone, CO, 2006.

[61] B. Macomber, A. B. Probe, R. M. Woollands, and J. L. Junkins, "Automated Tuning Parameter Selection for Orbit Propagation with Modified Chebyshev Pi-

card Iteration," in *25th AIAA/AAS Spaceflight Mechanics Meeting*, Williamsburg, VA, 2015.

[62] M. Berry and L. Healy, "Comparison of Accuracy Assesment Techniques for Numerical Integration," in *13th AAS/AIAA Space Flight Mechanics Meeting*, Ponce, Puerto Rico, 2003.

[63] R. Woollands, A. Bani Younes, B. Macomber, A. Probe, D. Kim, and J. L. Junkins, "Validation of Accuracy and Efficiency of Long-Arc Orbit Propagation Using the Method of Manufactured Solutions and Round-Trip-Closure Method," in *Advanced Maui Optical and Space Surveillance Technologies Conference*, Maui, Hawaii, 2014.

[64] P. E. Nacozy, "The Use of Integrals in Numerical Integrations of the N-Body Problem," in *Gravitational N-Body Problem*, 1972, vol. 31 of *Astrophysics and Space Science Library*, pp. 153–164.

[65] A. J. Sinclair and J. E. Hurtado, "The Eleventh Motion Constant of the Two-Body Problem," *Celestial Mechanics and Dynamical Astronomy*, vol. 110, pp. 189–198, 2011.

[66] J. L. Junkins, I. D. Jacobson, and J. N. Blanton, "A Non-Linear Oscillator Analog of Rigid Body Dynamics," *Celestial Mechanics*, vol. 7, pp. 398–407, 1973.

[67] B. Macomber, R. Woollands, A. Probe, A. Bani Younes, and J. L. Junkins, "Modified Chebyshev Picard Iteration for Efficient Numerical Integration of ordinary Differential Equations," in *Advanced Maui Optical and Space Surveillance Technologies Conference,*, Maui, Hawaii, 2013.

[68] P. L. Chebyshev, "Théorie des Mécanismes Connus Sous le Nom de Parallélogrammes," *Mémoires des Savants étrangers prèsentès à l'Acadèmie de SaintPètersbourg*, vol. 7, pp. 539–586, 1857.

APPENDIX A

LASR SSA COMPUTE CLUSTER

Head Node:

- 2x Intel Xeon® Processor E5-2630 v2
  - 6 Cores
  - 2.6 GHz
  - 3.1 GHz Turbo Boost
- 64 GB DDR3 1600 Memory
- 500 GB Raid 5 Hard Drive
- 6x Gigabit Ethernet
- Redundant Power Supply with 3000VA UPS
- PCI Express 2.0 expansion slots for accelerator processors (GPUs)

Compute Nodes (x15):

- 2x Intel Xeon® Processor E5-2630 v2
  - 6 Cores
  - 2.6 GHz
  - 3.1 GHz Turbo Boost
- 64 GB DDR3 1600 Memory
- 500 GB Hard Drive
- 6x Gigabit Ethernet

- PCI Express 2.0 expansion slots for accelerator processors (GPUs)

Software:

- Centos 6.5 Operating System - an open-source cousin of Red Hat Linux
- MPICH 3 Message Passing Interface
- Slurm (Simple Linux Utility for Resource Management)
- GNU C/C++/Fortran compilers
- Locally hosted yum package manager repository

The ideal computational capability of the LASR SSA Compute Cluster is roughly 2 TFLOPS (16 Nodes * 2 Sockets/Node * 6 Cores/Socket * 2.6 GHz clock cycle * 4 FLOPS/cycle = 1.997TFLOPS).

(a) Cluster front view. (b) Cluster rear view.

Figure A.1: LASR SSA Compute Cluster.

APPENDIX B

PICARD ITERATION

B.1 Analytic Picard Iteration Example

Consider the scalar, linear Ordinary Differential Equation

$$\dot{x} = 2x \tag{B.1}$$

with the initial condition $x(t_0) = x(0) = 1$. The differential equation has an analytic solution of the form

$$x(t) = ce^{2t} \tag{B.2}$$

which, by applying the given boundary condition, reduces to

$$x(t) = e^{2t} \tag{B.3}$$

Ignoring the analytic solution for the moment, we will rearrange the differential equation into an integral equation. We make use of the fact that for a general differential equation $\dot{x} = f(t, x(t))$:

$$\dot{x} = \frac{dx}{dt} = f(t, x(t)) \tag{B.4}$$

Thus, without approximation, the equation may be rewritten as

$$x(t) = x(t_0) + \int_{t_0}^{t} f(\tau, x(\tau))d\tau \tag{B.5}$$

Picard iteration allows us to iteratively improve our estimate of the true solution to the ODE by recursively evaluating the integral expression

$$x^i(t) = x(t_0) + \int_{t_0}^{t} f(\tau, x^{i-1}(\tau))d\tau \; i = 1, 2, ... \tag{B.6}$$

In this case $f(\tau, x^{i-1}(\tau)) = 2x^{i-1}$, as defined by the differential equation in Equation B.1.

We choose an initial estimate of the solution $x^0(t)$ equal to the boundary condition $x(t_0)$, which gives us

$$x^0(t) = 1 \tag{B.7}$$

Subsequent Picard iterations, we find the i^{th} estimate by evaluating the integrand with the $(i-1)^{th}$ estimate. The first iteration provides

$$x^1(t) = 1 + \int_0^t 2[1]d\tau \tag{B.8a}$$

$$x^1(t) = 1 + 2t \tag{B.8b}$$

The second iteration gives

$$x^2(t) = 1 + \int_0^t 2[1 + 2\tau]d\tau \tag{B.9a}$$

$$x^2(t) = 1 + 2t + \frac{4t^2}{2} \tag{B.9b}$$

The third iteration gives

$$x^3(t) = 1 + \int_0^t 2\left[1 + 2\tau + \frac{4t^2}{2}\right]d\tau \tag{B.10a}$$

$$x^3(t) = 1 + 2t + \frac{4t^2}{2} + \frac{8t^3}{6} \tag{B.10b}$$

After just three iterations a clear pattern has emerged:

$$x^3(t) = 1 + \frac{(2t)}{1} + \frac{(2t)^2}{2 \cdot 1} + \frac{(2t)^3}{3 \cdot 2 \cdot 1} \tag{B.11}$$

Thus, the pattern, when extrapolated to the n^{th} estimate, is

$$x^n(t) = \sum_{n=0}^{\infty} \frac{(2t)^n}{n!} \tag{B.12}$$

which is simply the Maclaurin series expansion of the analytic solution $x(t) = e^{2t}$.

Of course, we do not need Picard iteration to verify the known solution of a linear differential equation. It is a comforting validation however.

APPENDIX C

CHEBYSHEV POLYNOMIALS

C.1 Some Useful Properties

Chebyshev polynomials are an orthogonal basis set developed by the Russian mathematician Pafnuty Lvovich Chebyshev in 1857 [68]. Chebyshev polynomials of the first kind T_n exist on the domain and range $[-1, 1]$, and Chebyshev polynomials of the second kind U_n exist on the domain $[-1, 1]$ but are unbounded in range. The first six Chebyshev polynomials T_0 through T_5 are shown in Figure C.1[1]. Throughout this document it is inferred that references to Chebyshev polynomials means Chebyshev polynomials of the first kind. Derivations of the formulas that follow may be found in Fox and Parker [40].

Chebyshev polynomials may be generated explicitly for $\tau \in [-1, 1]$ by the trigonometric function

$$T_n(\tau) = -\cos(n \arccos(\tau)) \tag{C.1}$$

or recursively by the recurrence relation

$$T_0(\tau) = 1 \tag{C.2a}$$

$$T_1(\tau) = \tau \tag{C.2b}$$

$$T_{n+1}(\tau) = 2\tau T_n(\tau) - T_{n-1}(\tau) \tag{C.2c}$$

Within their domain of existence, the Chebyshev polynomials are orthogonal with

[1]Image credit - Wikimedia Commons / Public Domain.

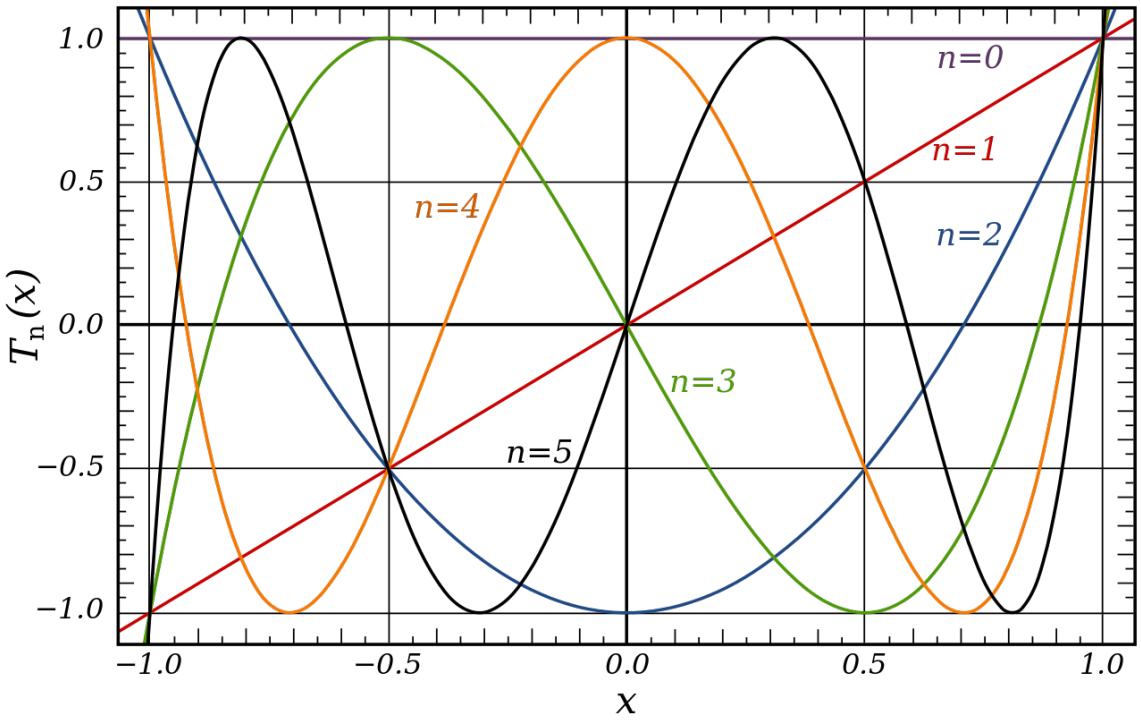


Figure C.1: The first six Chebyshev polynomials of the first kind: T_0 through T_5.

respect to the weight function

$$\omega(\tau) = \frac{1}{\sqrt{1-\tau^2}} \tag{C.3}$$

Using a continuous inner product, the orthogonality condition is

$$\int_{-1}^{1} \omega(\tau) T_n(\tau) T_m(\tau)\, d\tau = \begin{cases} 0 & : n \neq m \\ \pi & : n = m = 0 \\ \pi/2 & : n = m \neq 0 \end{cases} \tag{C.4}$$

A convenient discrete orthogonality property exists when sampling the Chebyshev polynomials at either their extrema or their zeros. The $M+1$ extrema of T_N are

located at the Chebyshev-Gauss-Lobatto (CGL) nodes

$$\tau_j = -\cos\left(\frac{\pi}{M}j\right),\ j = 0, 1, 2, ..., M \tag{C.5}$$

In the case that the number of CGL sample points is equal to the Chebyshev order of approximation ($M = N$), then the discrete orthogonality condition is

$$\sum_{j=0}^{M}{}'' \omega_j T_n(\tau_j) T_m(\tau_j) = \begin{cases} 0 & : n \neq m \\ M & : n = m = 0 \\ M/2 & : n = m \neq 0 \end{cases} \tag{C.6}$$

where the discrete weight function ω_j varies with node number j ($\omega_0 = \omega_N = \frac{1}{2}$, else $\omega_j = 1$). This varying weight function is denoted in Fox and Parker, and here henceforth, by a ($''$) symbol within the summation operator to indicate that the first and last terms have a $\frac{1}{2}$ multiplying them, while the rest of the terms have a 1 multiplying them. Similarly, a ($'$) within the summation is a notation that indicates that the first term in the summation is multiplied by $\frac{1}{2}$ while the rest of the terms are multiplied by 1.

Depending upon the order of Chebyshev approximation N, and the number of CGL sample points M (as defined by Equation C.5), discrete function approximation with the orthogonal Chebyshev polynomial basis set is performed using either a least squares operation ($M > N$), a direct interpolation formula ($M = N$), or a minimum norm operation ($M < N$). Because we are free to choose the number of sample points within the MCPI method, we choose to avoid the minimum norm formulation and instead make use of the direct interpolation or least squares methods.

Using $M > N$ samples at the CGL nodes, the N^{th}-order discrete least squares

"best" approximation $p_N(x(\tau))$ of the function $g(x(\tau))$ is given by

$$p_N(x(\tau)) \approx g(x(\tau)) = \sum_{k=0}^{N}{}' F_k T_k(x) \tag{C.7a}$$

$$F_k = \frac{2}{M} \sum_{j=0}^{M}{}'' g(x(\tau_j)) T_k(\tau_j) \tag{C.7b}$$

Thus approximated, the residual error $e_N(x) = g(x) - p_N(x)$ satisfies the discrete least squares criterion

$$S = \sum_{j=0}^{M}{}'' e_N^2(x(\tau_j)) = minimum \tag{C.8}$$

with a minimum residual error bound of

$$S_{min} = \sum_{j=0}^{M}{}'' \left\{ g^2(x(\tau_j)) - \sum_{k=0}^{N} F_k^2 T_k^2(\tau_j) \right\} \tag{C.9}$$

Choosing to use $M = N$ CGL samples, the direct interpolation formula provides the function

$$p_N(x(\tau)) = \sum_{k=0}^{N}{}'' F_k T_k(x) \tag{C.10a}$$

$$F_k = \frac{2}{M} \sum_{j=0}^{M}{}'' g(x(\tau_j)) T_k(\tau_j) \tag{C.10b}$$

which is the polynomial of degree N that fits the function $g(x(\tau))$ *exactly* at the CGL nodes. Note that the only difference between the least-squares formulation and the exact interpolation formulation is the extra $\frac{1}{2}$ term at the end of the series in Equation C.7a when compared to Equation C.10a. The least-squares error boundary of Equation C.9 provides an explicit measure of the accuracy of the approximation.

The interpolation formula is an exact fit at the CGL nodes, but no guarantee of the solution quality is provided in between the nodes. We will make use of both the least-squares and interpolation formulations within MCPI.

Derivatives and integrals of a sequence of Chebyshev polynomials are defined recursively in terms of the polynomials themselves. The indefinite integral of Chebyshev polynomials is defined for $n > 1$ by

$$\int T_n(\tau)d\tau = \frac{1}{2}\left(\frac{T_{n+1}(\tau)}{n+1} - \frac{T_{n-1}(\tau)}{n-1}\right) \tag{C.11}$$

and for $n \leq 1$ by

$$\int T_0(\tau)d\tau = T_1(\tau) \tag{C.12a}$$

$$\int T_1(\tau)d\tau = \frac{1}{2}T_2(\tau) + \frac{1}{2}T_0(\tau) \tag{C.12b}$$

APPENDIX D

MCPI TERM-BY-TERM DERIVATION

D.1 First-Order MCPI

In this appendix, a term-by-term explicit derivation of the first-order MCPI formulation is given. The system states will be approximated with 5^{th}-order ($N = 5$) Chebyshev polynomials, and it is shown how to generalize the result to arbitrary N. This section serves to fill in the gaps between the equations presented in Section 1.4.2, and to clean up a few issues in the existing literature.[1] In Section D.1.1 the core equations of MCPI are derived, and in Section D.1.2 the vector/matrix MCPI framework is derived from the core equations.

As described in Section 1.4.1, the independent time variable $t_0 \leq t \leq t_f$ is transformed to a new variable τ, which is defined over the valid domain of the Chebyshev polynomials $-1 \leq \tau \leq 1$. The original ODE is rewritten as an integral equation. Scaling the dynamics by the transformed variable τ, as described in Section 1.4.2, yields a Picard iteration formula for the transformed system given by Equation 1.10 (and repeated here for convenience):

$$\boldsymbol{x}^i(\tau) = \boldsymbol{x_0} + \int_{-1}^{\tau} \boldsymbol{g}(s, \boldsymbol{x}^{i-1}(s))ds \quad i = 1, 2, ... \tag{D.1}$$

[1] Credit is due to Robyn Woollands, upon whose notes this section is based. This detailed derivation has not been previously published, Robyn has organized it into an internal report for training future generations of MCPI students.

D.1.1 Traditional MCPI

The equations of MCPI are derived for a 5^{th} order Chebyshev polynomial approximation of the system states. For notational simplicity we shall consider a scalar ODE system. MCPI is applied to vector ODE systems by replacing these derived scalar Chebyshev coefficients, system states, and integrand functions with their vector counterparts.

Chebyshev polynomials are used to approximate the i^{th} Picard estimate of the system states on the left hand side of Equation D.1 (hereafter denoted **L.H.S.**) as

$$\begin{aligned} x^i(\tau_j) &\approx \sum_{k=0}^{N}{}'' \beta_k^i T_k(\tau_{\mathbf{j}}) \\ &= \frac{1}{2}\beta_0^i T_0(\tau_j) + \beta_1^i T_1(\tau_j) + \beta_2^i T_2(\tau_j) + ... + \frac{1}{2}\beta_N^i T_N(\tau_j) \end{aligned} \tag{D.2}$$

A separate Chebyshev polynomial sequence is used to approximate the *integrand* on the right hand side of Equation D.1 (hereafter denoted **R.H.S.**) as

$$\begin{aligned} g(s_j, x^{i-1}(s_j)) &\approx \sum_{k=0}^{N-1}{}' F_k^{i-1} T_k(s_j) \\ &= \frac{1}{2}F_0^{i-1} T_0(s_j) + F_1^{i-1} T_1(s_j) + F_2^{i-1} T_2(s_j) + ... + F_{N-1}^{i-1} T_{N-1}(s_j) \end{aligned} \tag{D.3}$$

Notice that the summation in Equation D.3 ranges from $k = 0$ to $N - 1$, whereas the summation in Equation D.2 ranges from $k = 0$ to N. We have specified that an $N = 5$ (5^{th}-order) Cheybshev approximation of the system states is desired. The **R.H.S.** approximation in Equation D.3 is within an integral sign, which means that the Cheybyshev order of approximation will increase by one, as shown in the analytic Chebyshev polynomial integral formula of Equation C.11. Therefore, by approximating the integrand with a Chebyshev series of a single degree lower, we will

end up with an equal order approximation after the **R.H.S.** is analytically integrated. We will use the same number of CGL sample points M (as given by Equation C.5) for both approximations, equal to the order of the Chebyshev fit of the states ($M = N$). This means that the **R.H.S.** uses the least squares fit method of Equations C.7, whereas the **L.H.S.** uses the interpolation fit method of Equations C.10, which is the reason for the 1/2 multiplying the last term on the **L.H.S.** but not the **R.H.S.**.

During the $(i-1)^{th}$ Picard iteration, the coefficients of the integrand approximation may be calculated directly using the least squares Chebyshev approximation method of Equation C.7b as

$$F_k^{i-1} = \frac{2}{M} \sum_{j=0}^{M}{}'' g(s_j, x^{i-1}(s_j)) T_k(s_j) \tag{D.4}$$

after having evaluated the integrand function $g(s_j, x^{i-1}(s_j))$ at the current best estimate of the system state x^{i-1}. Combining Equations D.2 and D.3, Picard iterations take the form

$$x^i(\tau) = \sum_{k=0}^{N}{}'' \beta_k^i T_k(\tau) = x(-1) + \int_{-1}^{\tau} \left[\sum_{k=0}^{N-1}{}' F_k^{i-1} T_k(s) \right] ds \tag{D.5}$$

where the $\boldsymbol{x_0}$ initial condition term in Equation D.1 is replaced by $x(-1)$ since the time has been linearly scaled to the $[-1, 1]$ domain where the Chebyshev polynomials exist. Upon integrating the **R.H.S.**, the process of Picard iteration is thus reduced to equating coefficients of like index basis functions across the equal sign in the above equation, obtaining the unknown state Chebyshev coefficients β_k^i in terms of the known integrand coefficients F_k^{i-1}. Expanding the expressions on both sides of the equation and grouping in terms of the k^{th} Chebyshev basis function will bring the desired result. Note that the only approximation is the fit of the integrand, there

Degree	Coefficient
τ^0	$\frac{1}{2}\beta_0 - \beta_2 + \beta_4$
τ^1	$\beta_1 - 3\beta_3 + \frac{5}{2}\beta_5$
τ^2	$2\beta_2 - 8\beta_4$
τ^3	$4\beta_3 - 10\beta_5$
τ^4	$8\beta_4$
τ^5	$8\beta_5$

Table D.1: State coefficients (L.H.S.) of first-order MCPI for $N = M = 5$.

is no Taylor series-like local linearization required to obtain the updated Chebyshev series for $x^i(\tau)$ from the $(i-1)^{th}$ acceleration Chebyshev series.

Focusing first on the **L.H.S.**, we substitute the expressions for the Chebyshev coefficients T_k as

$$x^i(\tau) = \frac{1}{2}\beta_0 + \beta_1(\tau) + \beta_2(2\tau^2 - 1) + \beta_3(4\tau^3 - 3\tau) + \\ + \beta_4(8\tau^4 - 8\tau^2 + 1) + \frac{1}{2}\beta_5(16\tau^5 - 20\tau^3 + 5\tau) = R.H.S. \tag{D.6}$$

Collecting the sets of coefficients modifying each power of τ yields

$$x^i(\tau) = \left[\frac{1}{2}\beta_0 - \beta_2 + \beta_4\right]\tau^0 + \left[\beta_1 - 3\beta_3 + \frac{5}{2}\beta_5\right]\tau^1 + \\ + \left[2\beta_2 - 8\beta_4\right]\tau^2 + \left[4\beta_3 - 10\beta_5\right]\tau^3 + \left[8\beta_4\right]\tau^4 + \left[8\beta_5\right]\tau^5 \tag{D.7}$$

The coefficients of each power of τ in the above equation are summarized in Table D.1.

Turning now to the **R.H.S.**, we find the expression

$$L.H.S. = x^i(\tau) = x(-1) + \frac{1}{2}F_0\int_{-1}^{\tau} T_0(s)ds + F_1\int_{-1}^{\tau} T_1(s)ds + \\ + F_2\int_{-1}^{\tau} T_2(s)ds + F_3\int_{-1}^{\tau} T_3(s)ds + \ F_4\int_{-1}^{\tau} T_4(s)ds \tag{D.8}$$

Making use of the analytic Chebyshev polynomial integration formula from Equation C.11, this may be written as

$$x(\tau) = x(-1) + \frac{1}{2}F_0 \int_{-1}^{\tau} 1ds + F_1 \int_{-1}^{\tau} sds + \frac{1}{2}F_2 \left[\frac{T_3(s)}{3} - \frac{T_1(s)}{1}\right]\Bigg|_{-1}^{\tau} + \\ + \frac{1}{2}F_3 \left[\frac{T_4(s)}{4} - \frac{T_2(s)}{2}\right]\Bigg|_{-1}^{\tau} + \frac{1}{2}F_4 \left[\frac{T_5(s)}{5} - \frac{T_3(s)}{3}\right]\Bigg|_{-1}^{\tau} \tag{D.9}$$

The first two terms do not need to be evaluated using the analytic Chebyshev integration formula, it is simpler to integrate them directly ($T_0(s) = 1$ and $T_1(s) = s$). By inspection of Figure C.1 it can be seen that

$$T_k(-1) = \begin{cases} -1 & k \text{ odd} \\ \\ +1 & k \text{ even} \end{cases} \tag{D.10}$$

Evaluating the first two integrals and substituting the Chebyshev polynomials $T_1(s)$ through $T_5(s)$ evaluated at $s = \tau$ and $s = -1$ for the remaining terms gives us

$$\begin{aligned} x(\tau) =& x(-1) + \frac{1}{2}F_0 \left[\tau + 1\right] + \frac{1}{2}F_1 \left[\tau^2 - 1\right] + \\ &+ \frac{1}{2}F_2 \left[\frac{4\tau^3 - 3\tau + 1}{3} - \frac{\tau + 1}{1}\right] + \frac{1}{2}F_3 \left[\frac{8\tau^4 - 8\tau^2}{4} - \frac{2\tau^2 - 2}{2}\right] \\ &+ \frac{1}{2}F_4 \left[\frac{16\tau^5 - 20\tau^3 + 5\tau + 1}{5} - \frac{4\tau^3 - 3\tau + 1}{3}\right] \end{aligned} \tag{D.11}$$

Again collecting the sets of coefficients modifying each power of τ yields the expres-

sion

$$x(\tau) = \left[x(-1) + \frac{1}{2}F_0 - \frac{1}{2}F_1 - \frac{1}{3}F_2 + \frac{1}{2}F_3 - \frac{1}{15}F_4\right]\tau^0 + \tag{D.12}$$
$$+ \left[\frac{1}{2}F_0 - F_2 + F_4\right]\tau^1 + \left[\frac{1}{2}F_1 - \frac{3}{2}F_3\right]\tau^2 + \left[\frac{2}{3}F_2 - \frac{8}{3}F_4\right]\tau^3$$
$$+ \left[F_3\right]\tau^4 + \left[\frac{8}{5}F_4\right]\tau^5$$

The coefficients of each power of τ in the above equation are summarized in Table D.2.

Degree	**Coefficient**
τ^0	$x(-1) + \frac{1}{2}F_0 - \frac{1}{2}F_1 - \frac{1}{3}F_2 + \frac{1}{2}F_3 - \frac{1}{15}F_4$
τ^1	$\frac{1}{2}F_0 - F_2 + F_4$
τ^2	$\frac{1}{2}F_1 - \frac{3}{2}F_3$
τ^3	$\frac{2}{3}F_2 - \frac{8}{3}F_4$
τ^4	F_3
τ^5	$\frac{8}{5}F_4$

Table D.2: Integrand coefficients (R.H.S.) of first-order MCPI for $N = M = 5$.

Having solved for the coefficients modifying each power of τ on both sides of the equation, we may now solve for the unknown state coefficients β_k^i of Table D.1 in terms of the known integrand coefficients F_k^i from Table D.2. Starting first with the coefficients multiplying τ^5 on both sides we find

$$8\beta_5 = \frac{8}{5}F_4 \tag{D.13a}$$
$$\beta_5 = \frac{1}{5}F_4 \tag{D.13b}$$

or more generally,

$$\beta_N = \frac{1}{N}F_{N-1} \tag{D.14}$$

Next, the coefficients for τ^4 give

$$8\beta_4 = F_3 \tag{D.15a}$$

$$\beta_4 = \frac{1}{8}F_3 \tag{D.15b}$$

or more generally,

$$\beta_{N-1} = \frac{1}{2(N-1)}F_{N-2} \tag{D.16}$$

Equating the coefficients for τ^3 gives us

$$4\beta_3 - 10\beta_5 = \frac{2}{3}F_2 - \frac{8}{3}F_4 \tag{D.17}$$

Substituting the relationship from Equation D.13b allows us to find

$$4\beta_3 - 10\left(\frac{F_4}{5}\right) = \frac{2}{3}F_2 - \frac{8}{3}F_4 \tag{D.18a}$$

$$\beta_3 = \frac{1}{6}\left(F_2 - F_4\right) \tag{D.18b}$$

or, written generally,

$$\beta_k = \frac{1}{2k}(F_{k-1} - F_{k+1}) \tag{D.19}$$

It it easily verified by the same procedure that the general expression for β_k in Equation D.19 holds for β_2 and β_1 as well.

Substituting $\tau = -1$ in Equation D.7 and D.12 and setting the two equal to one another yields the expression for β_0. Starting with Equation D.7 we find

$$\frac{1}{2}\beta_0 - \beta_1 + \beta_2 - \beta_3 + \beta_4 - \frac{1}{2}\beta_5 = R.H.S. \tag{D.20}$$

In Equation D.12, all the terms cancel out yielding the trivial result

$$L.H.S. = x(-1) \tag{D.21}$$

Equating the above two expressions and solving for coefficient β_0 yields

$$\beta_0 = 2x(-1) + 2(\beta_1 - \beta_2 + \beta_3 - \beta_4) + \beta_5 \tag{D.22}$$

From which we may deduce the general expression

$$\beta_0 = 2x_0 + 2\sum_{k=1}^{N-1}\left[(-1)^{k+1}\beta_k\right] + \left[(-1)^{k+1}(-1)^{N+1}\beta_N\right] \tag{D.23}$$

Expanding the expression for β_0 in Equation D.22 in terms of the other β_k for which we have just solved, we find the expression

$$\beta_0 = 2x_0 + 2\left[\frac{1}{2}(F_0 - F_2) - \frac{1}{4}(F_1 - F_3) + \frac{1}{6}(F_2 - F_4) - \frac{1}{8}(F_3)\right] + \frac{1}{5}(F_4) \tag{D.24a}$$

$$\beta_0 = 2x_0 + 1(F_0) - \frac{1}{2}(F_1) - \frac{2}{3}(F_2) + \frac{1}{4}(F_3) - \frac{2}{15}(F_4) \tag{D.24b}$$

We will utilize Equation D.24b in the next section.

Summarizing the above section, we have found the general expressions for Picard iteration updates of the form of Equation D.5. The $(i-1)^{th}$ approximation of the integrand coefficients is found by performing an $(N-1)^{th}$ order least squares Chebyshev approximation of the integrand $g(s_j, x^{i-1}(s_j))$, evaluated along the current best estimate of the system state trajectory x^{i-1}, yielding the integrand coefficients F_k^{i-1}. The Picard iteration consists of calculating the i^{th} approximation of the state

coefficients β_k^i using the expressions

$$\beta_N = \frac{1}{N} F_{N-1} \tag{D.25a}$$

$$\beta_{N-1} = \frac{1}{2(N-1)} F_{N-2} \tag{D.25b}$$

$$\beta_k = \frac{1}{2k}(F_{k-1} - F_{k+1}) \ , \quad k = 1, 2, .., N-2 \tag{D.25c}$$

$$\beta_0 = 2x_0 + 2\sum_{k=1}^{N-1} \left[(-1)^{k+1}\beta_k\right] + \left[(-1)^{N+1}\beta_N\right] \tag{D.25d}$$

We mention that the above process is presented for the purpose of illustration. Identical expressions (without truncating to 5^{th} order) can be obtained by a more general process, using the analytic expressions for integration of Chebyshev polynomials as given by Equations C.11 and C.12 in the integrand on the **R.H.S.** of Equation D.5. Upon collecting terms and equating the coefficients of $T_k(\tau)$ on the **L.H.S.** to the corresponding coefficients on the **R.H.S.**, and imposing the initial boundary conditions at $\tau = -1$, Equations D.25 are established.

D.1.2 Vector/Matrix MCPI

The integrand of the ODE is approximated using the least squares Chebyshev approximation of Equation C.7b, repeated here for convenience:

$$\boldsymbol{F}_k = \frac{2}{M}\sum_{j=0}^{M}{}''\boldsymbol{g}(\boldsymbol{x}(\tau_j))T_k(\tau_j) \tag{D.26}$$

Note that, while the previous section derived the core MCPI expressions for a scalar system (state vector $\boldsymbol{x}$ reduces to a scalar: $x \in \mathbb{R}^1$), here we generalize to a state vector of arbitrary length $\boldsymbol{x} \in \mathbb{R}^n$. We have chosen to approximate the system states using an N^{th} order Chebyshev polynomial series, which means that the integrand must be approximated with an $(N-1)^{th}$ order series because of the increase of

order due to the analytic integration property of Chebyshev polynomials. Both approximations are performed with $M = N$ CGL sample points, which means the integrand fit uses the least squares formulation, and the state fit uses the interpolation formulation. The integrand fit of Equation D.26 may be written as a vector/matrix system:

$$F^{i-1} = \begin{bmatrix} (\boldsymbol{F}_0^{i-1})^T \\ (\boldsymbol{F}_1^{i-1})^T \\ \vdots \\ (\boldsymbol{F}_{N-2}^{i-1})^T \\ (\boldsymbol{F}_{N-1}^{i-1})^T \end{bmatrix} = \tag{D.27a}$$

$$= \begin{bmatrix} T_0(\tau_0) & T_0(\tau_1) & \dots & T_0(\tau_M) \\ T_1(\tau_0) & T_1(\tau_1) & \dots & T_1(\tau_M) \\ \vdots & \vdots & \dots & \vdots \\ T_{N-2}(\tau_0) & T_{N-2}(\tau_1) & \dots & T_{N-2}(\tau_M) \\ T_{N-1}(\tau_0) & T_{N-1}(\tau_1) & \dots & T_{N-1}(\tau_M) \end{bmatrix} \begin{bmatrix} \frac{1}{M} & & & & \\ & \frac{2}{M} & & & \\ & & \ddots & & \\ & & & \frac{2}{M} & \\ & & & & \frac{1}{M} \end{bmatrix} \begin{bmatrix} \boldsymbol{G}^{i-1}(\tau_0) \\ \boldsymbol{G}^{i-1}(\tau_1) \\ \vdots \\ \boldsymbol{G}^{i-1}(\tau_{M-1}) \\ \boldsymbol{G}^{i-1}(\tau_M) \end{bmatrix} \tag{D.27b}$$

or equivalently,

$$F^{i-1} = {}^{F}T^{T} V G(X^{i-1}) \tag{D.28}$$

F^{i-1} is an $(N \times n)$ matrix of integrand coefficients (n is the length of the state vector: $\boldsymbol{x} \in \mathbb{R}^n$), ${}^{F}T$ is an $(M+1) \times N$ matrix of Chebyshev polynomials evaluated at the CGL sample points (${}^{F}T^{T}$ is the transpose of ${}^{F}T$), and V is a $(M+1)\times(M+1)$ weight matrix. $G(X^{i-1})$ is an $(M+1) \times n$ matrix of the values from the evaluations of the

integrand approximation function at the $(i-1)^{th}$ estimate of the states $\boldsymbol{g}(\tau_j, \boldsymbol{x}^{i-1}(\tau_j))$

$$\boldsymbol{G}^{i-1}(\tau_j) = \boldsymbol{g}(\tau_j, \boldsymbol{x}^{i-1}(\tau_j))^T \tag{D.29a}$$

$$G(X^{i-1}) = matrix\{\boldsymbol{G}^{i-1}(\tau_0)\ ;\ ...\ ;\ \boldsymbol{G}^{i-1}(\tau_M)\} \tag{D.29b}$$

Grouping the state coefficients into an $(N+1) \times n$ matrix β^i, the relationships of the state coefficients from Equations D.25 may be written in vector/matrix form:

$$\beta^i = \begin{bmatrix} (\boldsymbol{\beta}_0^i)^T \\ (\boldsymbol{\beta}_1^i)^T \\ \vdots \\ (\boldsymbol{\beta}_k^i)^T \\ \vdots \\ (\boldsymbol{\beta}_{N-1}^i)^T \\ (\boldsymbol{\beta}_N^i)^T \end{bmatrix} = \begin{bmatrix} 2\boldsymbol{x}_\mathbf{0}^T + 2\sum_{k=1}^{N-1}\left[(-1)^{k+1}\boldsymbol{\beta}_k^i\right]^T + \left[(-1)^{N+1}\boldsymbol{\beta}_N^i\right]^T \\ \frac{1}{2}(\boldsymbol{F}_0^{i-1} - \boldsymbol{F}_2^{i-1})^T \\ \vdots \\ \frac{1}{2k}(\boldsymbol{F}_{k-1}^{i-1} - \boldsymbol{F}_{k+1}^{i-1})^T \\ \vdots \\ \frac{1}{2(N-1)}(\boldsymbol{F}_{N-2}^{i-1})^T \\ \frac{1}{N}(\boldsymbol{F}_{N-1}^{i-1})^T \end{bmatrix} \tag{D.30}$$

$$\beta^i = \begin{bmatrix} 2\boldsymbol{x}_\mathbf{0}^T \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} 1 & & & & & & \\ & \frac{1}{2} & & & & & \\ & & \ddots & & & & \\ & & & \frac{1}{2k} & & & \\ & & & & \ddots & & \\ & & & & & \frac{1}{2(N-1)} & \\ & & & & & & \frac{1}{N} \end{bmatrix} \begin{bmatrix} 1 & -\frac{1}{2} & S(1,3) & \dots & S(1,k) & \dots & S(1,N) \\ 1 & 0 & -1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & -1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} (\boldsymbol{F}_0^{i-1})^T \\ (\boldsymbol{F}_1^{i-1})^T \\ (\boldsymbol{F}_2^{i-1})^T \\ \vdots \\ (\boldsymbol{F}_{N-3}^{i-1})^T \\ (\boldsymbol{F}_{N-2}^{i-1})^T \\ (\boldsymbol{F}_{N-1}^{i-1})^T \end{bmatrix} \tag{D.31}$$

Equation D.31 may be written more compactly as

$$\beta^i = X_0 + RSF^{i-1} \tag{D.32}$$

where R is an $(N+1) \times (N+1)$ matrix, S is an $(N+1) \times N$ matrix, and X_0 is an $(N+1) \times n$ matrix that enforces the initial boundary upon the state coefficients. The top row of the S matrix has the generally defined value $S(1,k)$ for $k = 3, 4, ..., N$. From Equation D.24b in the previous section, we see empirically the values that $S(1,k)$ should take for the $N = 5$ Chebyshev approximation case:

$$S(1,3) = -\frac{2}{3} \tag{D.33a}$$

$$S(1,4) = +\frac{1}{4} \tag{D.33b}$$

$$S(1,5) = -\frac{2}{15} \tag{D.33c}$$

It may be verified that the generally defined value

$$S(1,k) \equiv (-1)^k \left(\frac{1}{k-2} - \frac{1}{k} \right) \tag{D.34}$$

satisfies the empirical requirements of Equations D.33.

Having defined the state coefficients β_i, the states themselves are given by the interpolation formulation Chebyshev series of Equation C.10a, repeated here in a form with modified notation matching that used throughout this section:

$$x^i(\tau) \approx \sum_{k=0}^{N}{}'' \beta_k^i T_k(\tau) \tag{D.35}$$

This equation may be expressed as

$$X^i = \begin{bmatrix} T_0(\tau_0) & T_1(\tau_0) & \dots & T_N(\tau_0) \\ T_0(\tau_1) & T_1(\tau_1) & \dots & T_N(\tau_1) \\ \vdots & \vdots & \dots & \vdots \\ T_0(\tau_M) & T_1(\tau_M) & \dots & T_N(\tau_M) \end{bmatrix} \begin{bmatrix} \frac{1}{2} & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & \frac{1}{2} \end{bmatrix} \begin{bmatrix} (\boldsymbol{\beta_0^i})^T \\ \vdots \\ (\boldsymbol{\beta_k^i})^T \\ \vdots \\ (\boldsymbol{\beta_N^i})^T \end{bmatrix} \tag{D.36}$$

or, in shorthand

$$X^i = {}^{\beta}T\,{}^{\beta}W\beta^i \tag{D.37}$$

${}^{\beta}W$ is an $(N+1)\times(N+1)$ diagonal weight matrix, and ${}^{\beta}T$ is a matrix of Chebyshev polynomials evaluated at CGL sampled times τ_j, in this case ${}^{\beta}T$ is of size $(M+1)\times(N+1)$. The somewhat cryptic notation is required since ${}^{F}T^{T}$ (used for approximating the integrand) is a different size than ${}^{\beta}T$. Since we have used $M=N$ (the number of CGL sample points is equal to the order of Chebyshev approximation), ${}^{\beta}T$ is a square, symmetric matrix. We have defined the i^{th} estimate of the state as a matrix by stacking the state estimate values for all $(M+1)$ sampled times τ_0 to τ_M as

$$X^i = matrix\{\boldsymbol{x}^i(\tau_0)^T \ ; \ \boldsymbol{x}^i(\tau_1)^T \ ; \ \dots \ ; \ \boldsymbol{x}^i(\tau_M)^T\} \tag{D.38}$$

where each $\boldsymbol{x}^i(\tau_j) = [x_1^i(\tau_j) \ , \ x_2^i(\tau_j) \ , \ \dots \ , \ x_n^i(\tau_j)]^T$ is an $(n\times 1)$ vector.

Summarizing the developments above, the process of Chebyshev approximation

and Picard iteration is

$$F^{i-1} = {}^{F}T^{T}VG(X^{i-1}) \tag{D.39a}$$

$$\beta^{i} = X_0 + RSF^{i-1} \tag{D.39b}$$

$$X^{i} = {}^{\beta}T^{\beta}W\beta^{i} \tag{D.39c}$$

Defining two constant matrices

$$C_\alpha = RS^{F}T^{T}V \tag{D.40a}$$

$$C_x = {}^{\beta}T^{\beta}W \tag{D.40b}$$

we may simplify the MCPI approximation and update steps to

$$X^{i} = C_x C_\alpha G(X^{i-1}) + C_x X_0 \tag{D.41}$$

D.2 Second-Order MCPI

In this section, an explicit derivation of MCPI for naturally second-order ODE systems is given. Similarly to Section D.1 for MCPI for first-order ODEs, the explicit derivation is performed for a low order ($N = 6$) Chebyshev approximation, and it is shown how to generalize the result to arbitrary N. In Section D.2.1 the core equations are derived, and in Section D.2.2 the vector/matrix framework is derived from the core equations.

As described in Section 1.4.1, the independent time variable $t_0 \leq t \leq t_f$ is transformed to a new variable τ, which is defined over the valid domain of the Chebyshev polynomials $-1 \leq \tau \leq 1$. The original ODE is rewritten as an integral equation. Scaling the dynamics by the transformed variable τ, as described in Section 1.4.2,

yields a Picard iteration formula for the transformed system given by Equation 1.36 (and repeated here for convenience):

$$\boldsymbol{v}^i(\tau) = \boldsymbol{v_0} + \int_{-1}^{\tau} \boldsymbol{g}(s, \boldsymbol{x}^{i-1}(s), \boldsymbol{v}^{i-1}(s))ds, \quad i = 1, 2, ... \tag{D.42}$$

and kinematic updates to the position states given by Equation 1.37:

$$\boldsymbol{x}^i(\tau) = \boldsymbol{x_0} + \int_{-1}^{\tau} \boldsymbol{v}^i(s)ds \tag{D.43}$$

D.2.1 Traditional MCPI

The equations of second-order MCPI are derived for a 6^{th}-order Cheybshev polynomial approximation of the system states. For illustration, we shall again consider a scalar ODE system, but the vector version of the equations is structurally identical. The integrand, the velocity level states, and the position level states will be approximated with a Chebyshev series, and the analytic integral Chebyshev polynomial property will be used to kinematically relate the approximations. We will use $M = N = 6$ CGL sample points for all approximations when we need to be specific for this demonstration.

Using the interpolation approximation formulation of Equation C.10, an N^{th}-order Chebyshev polynomial sequence is used to approximate the i^{th} Picard estimate of the system position states

$$\begin{aligned} x^i(\tau) &\approx \sum_{k=0}^{N}{}'' \alpha_k^i T_k(\tau) \\ &= \frac{1}{2}\alpha_0^i T_0(\tau) + \alpha_1^i T_1(\tau) + \alpha_2^i T_2(\tau) + ... + \frac{1}{2}\alpha_N^i T_N(\tau) \end{aligned} \tag{D.44}$$

The velocity states are approximated with an $(N-1)^{th}$-order Chebyshev series

using the least squares Chebyshev approximation formulation of Equation C.7

$$v^i(\tau) \approx \sum_{k=0}^{N-1}{}' \beta_k^i T_k(\tau) \tag{D.45}$$
$$= \frac{1}{2}\beta_0^i T_0(\tau) + \beta_1^i T_1(\tau) + \beta_2^i T_2(\tau) + ... + \beta_{N-1}^i T_{N-1}(\tau)$$

As discussed in the previous section, the summation in Equation D.45 is from $k = 0$ to $N - 1$ since the analytic integration property causes the Chebyshev order of approximation to increase by one, thus the reason for using the least square formulation for the velocity states instead of the interpolation formulation.

A separate Chebyshev polynomial sequence of order $(N - 2)$ is used to approximate the *integrand* on the right hand side of Equation D.42

$$g(s, x^{i-1}(s), v^{i-1}(s)) \approx \sum_{k=0}^{N-2}{}' F_k^{i-1} T_k(s) \tag{D.46}$$
$$= \frac{1}{2}F_0^{i-1} T_0(s) + F_1^{i-1} T_1(s) + F_2^{i-1} T_2(s) + ... + F_{N-2}^{i-1} T_{N-2}(s)$$

again using the least squares approximation formulation. As in first-order MCPI, the integrand coefficients are calculated by Equation D.4 after having evaluated the integrand function $g(s_j, x^{i-1}(s_j), v^{i-1}(s_j))$ at the current best estimate of the system position and velocity states (x^{i-1} and v^{i-1}). Combining Equations D.44 through D.46, updates to the integrand and state coefficients take the form

$$v^i(\tau) = \sum_{k=0}^{N-1}{}' \beta_k^i T_k(\tau) = v(-1) + \int_{-1}^{\tau} \left[\sum_{k=0}^{N-2}{}' F_k^{i-1} T_k(s) \right] ds \tag{D.47a}$$

$$x^i(\tau) = x(-1) + \int_{-1}^{\tau} v^i(s) ds \tag{D.47b}$$

$$x^i(\tau) = \sum_{k=0}^{N}{}'' \alpha_k^i T_k(\tau) = x(-1) + \int_{-1}^{\tau} \left[\sum_{k=0}^{N-1}{}' \beta_k^i T_k(s) \right] ds \tag{D.47c}$$

where $v(-1)$ and $x(-1)$ are the initial conditions of the velocity and position states. Equation D.47a is a Picard iteration, and Equations D.47b and D.47c are a kinematic update.

Beginning with Equation D.47a, we must solve for the unknown velocity state coefficients β_k^i in terms of the known integrand coefficients F_k^{i-1}. The expressions on the left-hand-side (**L.H.S.**) may be expanded as a power series as

$$v^i(\tau) = \frac{1}{2}\beta_0 + \beta_1(\tau) + \beta_2(2\tau^2 - 1) + \beta_3(4\tau^3 - 3\tau) + \\ + \beta_4(8\tau^4 - 8\tau^2 + 1) + \beta_5(16\tau^5 - 20\tau^3 + 5\tau) = R.H.S. \tag{D.48}$$

which is nearly identical to Equation D.6 except for the lack of a $\frac{1}{2}$ multiplying the last term. Collecting the sets of coefficients modifying each power of τ yields

$$v^i(\tau) = \left[\frac{1}{2}\beta_0 - \beta_2 + \beta_4\right]\tau^0 + \left[\beta_1 - 3\beta_3 + 5\beta_5\right]\tau^1 + \\ + \left[2\beta_2 - 8\beta_4\right]\tau^2 + \left[4\beta_3 - 20\beta_5\right]\tau^3 + \left[8\beta_4\right]\tau^4 + \left[16\beta_5\right]\tau^5 \tag{D.49}$$

The coefficients of each power of τ in the above equation are summarized in Table D.3.

Degree	**Coefficient**
τ^0	$\frac{1}{2}\beta_0 - \beta_2 + \beta_4$
τ^1	$\beta_1 - 3\beta_3 + 5\beta_5$
τ^2	$2\beta_2 - 8\beta_4$
τ^3	$4\beta_3 - 20\beta_5$
τ^4	$8\beta_4$
τ^5	$16\beta_5$

Table D.3: State coefficients (L.H.S.) of velocity update in second-order MCPI for $N = M = 6$.

Expanding the right-hand-side (**R.H.S**) of Equation D.47a yields

$$L.H.S. = v^i(\tau) = v(-1) + \frac{1}{2}F_0\int_{-1}^{\tau} T_0(s)ds + F_1\int_{-1}^{\tau} T_1(s)ds + \\ + F_2\int_{-1}^{\tau} T_2(s)ds + F_3\int_{-1}^{\tau} T_3(s)ds + \ F_4\int_{-1}^{\tau} T_4(s)ds \tag{D.50}$$

which is identical to Equation D.8 except that here the initial condition is $v(-1)$ instead of $x(-1)$ as it was in the first-order MCPI case. We may therefore conclude that the coefficients for **R.H.S** are the same as Table D.2, which is repeated in Table D.4 for convenience.

Degree	**Coefficient**
τ^0	$v(-1) + \frac{1}{2}F_0 - \frac{1}{2}F_1 - \frac{1}{3}F_2 + \frac{1}{2}F_3 - \frac{1}{15}F_4$
τ^1	$\frac{1}{2}F_0 - F_2 + F_4$
τ^2	$\frac{1}{2}F_1 - \frac{3}{2}F_3$
τ^3	$\frac{2}{3}F_2 - \frac{8}{3}F_4$
τ^4	F_3
τ^5	$\frac{8}{5}F_4$

Table D.4: Integrand coefficients (R.H.S.) of velocity update in second-order MCPI for $N = M = 6$.

Having solved for the coefficients modifying each power of τ on both sides of the equation, we may now solve for the unknown velocity state coefficients β_k^i of Table D.3 in terms of the known integrand coefficients F_k^i from Table D.4. Starting first with the coefficients multiplying τ^5 on both sides we find

$$16\beta_5 = \frac{8}{5}F_4 \tag{D.51a}$$

$$\beta_5 = \frac{1}{10}F_4 \tag{D.51b}$$

or more generally,

$$\beta_{N-1} = \frac{1}{2(N-1)} F_{N-2} \tag{D.52}$$

Next, the coefficients for τ^4 give

$$8\beta_4 = F_3 \tag{D.53a}$$

$$\beta_4 = \frac{1}{8} F_3 \tag{D.53b}$$

or more generally,

$$\beta_{N-2} = \frac{1}{2(N-2)} F_{N-3} \tag{D.54}$$

Equating the coefficients for τ^3 gives us

$$4\beta_3 - 20\beta_5 = \frac{2}{3} F_2 - \frac{8}{3} F_4 \tag{D.55}$$

Substituting the relationship from Equation D.51b allows us to find

$$4\beta_3 - 20\left(\frac{F_4}{10}\right) = \frac{2}{3} F_2 - \frac{8}{3} F_4 \tag{D.56a}$$

$$\beta_3 = \frac{1}{6}\left(F_2 - F_4\right) \tag{D.56b}$$

or, written generally,

$$\beta_k = \frac{1}{2k}(F_{k-1} - F_{k+1}) \tag{D.57}$$

It it easily verified by the same procedure that the general expression for β_k in Equation D.57 holds for β_2 and β_1 as well.

To find the expression for β_0 is the same procedure as in Section D.1.1, namely substituting $\tau = -1$ into both sides of the equations. As in the first order section,

doing so on the **R.H.S** causes all F_k terms to cancel out, leaving the trivial solution

$$L.H.S. = v(-1) \tag{D.58}$$

On the **L.H.S**, upon substituting $\tau = -1$ in Equation D.49 we find the expression

$$\frac{1}{2}\beta_0 - \beta_1 + \beta_2 - \beta_3 + \beta_4 - \beta_5 = R.H.S. \tag{D.59}$$

Equating these two expressions and solving for β_0 gives the result

$$\beta_0 = 2v_0 + 2(\beta_1 - \beta_2 + \beta_3 - \beta_4 + \beta_5) \tag{D.60}$$

which may be expressed generally as

$$\beta_0 = 2v_0 + 2\sum_{k=1}^{N-1}\left[(-1)^{k+1}\beta_k\right] \tag{D.61}$$

Therefore we have solved for the unknown velocity state coefficients β_k^i in terms of the known integrand coefficients F_k^{i-1} for the Picard iteration of Equation D.47a. Expanding the expression for β_0 in Equation D.61 in terms of the other β_k for which we have just solved, we find the expression

$$\beta_0 = 2v_0 + 2\left[\frac{1}{2}(F_0 - F_2) - \frac{1}{4}(F_1 - F_3) + \frac{1}{6}(F_2 - F_4) - \frac{1}{8}(F_3) + \frac{1}{10}(F_4)\right] \tag{D.62a}$$

$$\beta_0 = 2v_0 + 1(F_0) - \frac{1}{2}(F_1) - \frac{2}{3}(F_2) + \frac{1}{4}(F_3) - \frac{2}{15}(F_4) \tag{D.62b}$$

Equation D.62b will be utilized in the next section.

A similar process is required to solve for the position state coefficients α_k in terms of the velocity state coefficients in the kinematic update of Equation D.47c.

Expanding the **L.H.S** of Equation D.47c provides

$$x^i(\tau) = \frac{1}{2}\alpha_0 + \alpha_1(\tau) + \alpha_2(2\tau^2 - 1) + \alpha_3(4\tau^3 - 3\tau) + \alpha_4(8\tau^4 - 8\tau^2 + 1) + \tag{D.63}$$
$$+ \alpha_5(16\tau^5 - 20\tau^3 + 5\tau) + \frac{1}{2}\alpha_6(32\tau^6 - 48\tau^4 + 18\tau^2 - 1) = R.H.S.$$

Grouping by coefficients modifying powers of τ gives

$$x^i(\tau) = \left[\frac{1}{2}\alpha_0 - \alpha_2 + \alpha_4 - \frac{1}{2}\alpha_6\right]\tau^0 + \left[\alpha_1 - 3\alpha_3 + 5\alpha_5\right]\tau^1 + \tag{D.64}$$
$$+ \left[2\alpha_2 - 8\alpha_4 + 9\alpha_6\right]\tau^2 + \left[4\alpha_3 - 20\alpha_5\right]\tau^3 + \left[8\alpha_4 - 24\alpha_6\right]\tau^4 +$$
$$+ \left[16\alpha_5\right]\tau^5 + \left[16\alpha_6\right]\tau^6$$

The coefficients of each power of τ in the above equation are summarized in Table D.5.

Degree	**Coefficient**
τ^0	$\frac{1}{2}\alpha_0 - \alpha_2 + \alpha_4 - \frac{1}{2}\alpha_6$
τ^1	$\alpha_1 - 3\alpha_3 + 5\alpha_5$
τ^2	$2\alpha_2 - 8\alpha_4 + 9\alpha_6$
τ^3	$4\alpha_3 - 20\alpha_5$
τ^4	$8\alpha_4 - 24\alpha_6$
τ^5	$16\alpha_5$
τ^6	$16\alpha_6$

Table D.5: State coefficients α_k (L.H.S.) of position update in second-order MCPI for $N = M = 6$.

Expanding the (**R.H.S**) of Equation D.47c yields

$$L.H.S. = x^i(\tau) = x(-1) + \frac{1}{2}\beta_0 \int_{-1}^{\tau} T_0(s)ds + \beta_1 \int_{-1}^{\tau} T_1(s)ds + \beta_2 \int_{-1}^{\tau} T_2(s)ds + \beta_3 \int_{-1}^{\tau} T_3(s)ds + \beta_4 \int_{-1}^{\tau} T_4(s)ds + \beta_5 \int_{-1}^{\tau} T_5(s)ds \tag{D.65}$$

Making use of the analytic Chebyshev polynomial integration formula from Equation C.11, this may be written as

$$\begin{aligned} x(\tau) = x(-1) + \frac{1}{2}\beta_0 \int_{-1}^{\tau} 1ds + \beta_1 \int_{-1}^{\tau} sds + \frac{1}{2}\beta_2 \left[\frac{T_3(s)}{3} - \frac{T_1(s)}{1}\right]\Bigg|_{-1}^{\tau} + \\ + \frac{1}{2}\beta_3 \left[\frac{T_4(s)}{4} - \frac{T_2(s)}{2}\right]\Bigg|_{-1}^{\tau} + \frac{1}{2}\beta_4 \left[\frac{T_5(s)}{5} - \frac{T_3(s)}{3}\right]\Bigg|_{-1}^{\tau} \\ + \frac{1}{2}\beta_5 \left[\frac{T_6(s)}{6} - \frac{T_4(s)}{4}\right]\Bigg|_{-1}^{\tau} \end{aligned} \tag{D.66}$$

Substituting the expressions of the Chebyshev polynomials $T_k(s)$, evaluating at the integration bounds $s = -1$ and $s = \tau$, and grouping by powers of τ, we find the resultant expression

$$\begin{aligned} x^i(\tau) = x(-1) + \left[\frac{1}{2}\beta_0 - \frac{1}{2}\beta_1 - \frac{1}{3}\beta_2 + \frac{1}{2}\beta_3 - \frac{1}{15}\beta_4 - \frac{1}{6}\beta_5\right]\tau^0 + \\ + \left[\frac{1}{2}\beta_0 - \beta_2 + \beta_4\right]\tau^1 + \left[\frac{1}{2}\beta_1 - \frac{3}{2}\beta_3 + \frac{5}{2}\beta_5\right]\tau^2 + \left[\frac{2}{3}\beta_2 - \frac{8}{3}\beta_4\right]\tau^3 + \\ + \left[\beta_3 - 5\beta_5\right]\tau^4 + \left[\frac{8}{5}\beta_4\right]\tau^5 + \left[\frac{8}{3}\beta_5\right]\tau^6 \end{aligned} \tag{D.67}$$

The coefficients of each power of τ in the above equation are summarized in Table D.6.

Solving for the unknown position state coefficients α_k in terms of the known velocity state coefficients β_k is accomplished by equating coefficients of equivalent degrees of τ between Table D.5 and D.6. Starting first with the coefficients multiplying τ^6

Degree	Coefficient
τ^0	$x(-1) + \frac{1}{2}\beta_0 - \frac{1}{2}\beta_1 - \frac{1}{3}\beta_2 + \frac{1}{2}\beta_3 - \frac{1}{15}\beta_4 - \frac{1}{6}\beta_5$
τ^1	$\frac{1}{2}\beta_0 - \beta_2 + \beta_4$
τ^2	$\frac{1}{2}\beta_1 - \frac{3}{2}\beta_3 + \frac{5}{2}\beta_5$
τ^3	$\frac{2}{3}\beta_2 - \frac{8}{3}\beta_4$
τ^4	$\beta_3 - 5\beta_5$
τ^5	$\frac{8}{5}\beta_4$
τ^6	$\frac{8}{3}\beta_5$

Table D.6: State coefficients β_k (R.H.S.) of position update in second-order MCPI for $N = M = 6$.

we find

$$16\alpha_6 = \frac{8}{3}\beta_5 \tag{D.68a}$$

$$\alpha_6 = \frac{1}{6}\beta_5 \tag{D.68b}$$

or more generally,

$$\alpha_N = \frac{1}{N}\beta_{N-1} \tag{D.69}$$

Next, the coefficients for τ^5 give

$$16\alpha_5 = \frac{8}{5}\beta_4 \tag{D.70a}$$

$$\alpha_5 = \frac{1}{10}\beta_4 \tag{D.70b}$$

or more generally,

$$\alpha_{N-1} = \frac{1}{2(N-1)}\beta_{N-2} \tag{D.71}$$

Equating the coefficients for τ^3 gives us

$$8\alpha_4 - 24\alpha_6 = \beta_3 - 5\beta_5 \tag{D.72}$$

Substituting the relationship from Equation D.68b allows us to find

$$\alpha_4 = \frac{1}{8}\left(\beta_3 - \beta_5\right) \tag{D.73}$$

or, written generally, we find the familiar form

$$\alpha_k = \frac{1}{2k}(\beta_{k-1} - \beta_{k+1}) \tag{D.74}$$

It it easily verified by the same procedure that the general expression for α_k in Equation D.74 holds for α_3 through α_1 as well.

Substituting $\tau = -1$ into Equation D.64 and D.67 and equating them yields the expression for α_0:

$$\alpha_0 = 2x_0 + 2(\alpha_1 - \alpha_2 + \alpha_3 - \alpha_4 + \alpha_5) - \alpha_6 \tag{D.75}$$

which may be written generally as

$$\alpha_0 = 2x_0 + 2\sum_{k=1}^{N-1}\left[(-1)^{k+1}\alpha_k\right] + \left[(-1)^{N+1}\alpha_N\right] \tag{D.76}$$

Expanding the expression for α_0 in Equation D.76 in terms of the other α_k for which we have just solved, we find the expression

$$\alpha_0 = 2x_0 + 2\left[\frac{1}{2}(\beta_0 - \beta_2) - \frac{1}{4}(\beta_1 - \beta_3) + \frac{1}{6}(\beta_2 - \beta_4) - \frac{1}{8}(\beta_3 - \beta_5) + \frac{1}{10}(\beta_4)\right] - \frac{1}{6}\beta_5 \tag{D.77a}$$

$$\alpha_0 = 2x_0 + 1(\beta_0) - \frac{1}{2}(\beta_1) - \frac{2}{3}(\beta_2) + \frac{1}{4}(\beta_3) - \frac{2}{15}(\beta_4) + \frac{1}{12}(\beta_5) \tag{D.77b}$$

Equation D.77b will be utilized in the next section.

Summarizing the above section, we have explicitly derived coefficient relationships for a kinematically consistent second-order MCPI algorithm of the form of Equations D.47. The integrand coefficients are approximated with an $(N-2)^{th}$-order Chebyshev least squares approximation, with coefficients F_0^{i-1} through F_{N-2}^{i-1}, using Equation C.7. The velocity states are approximated using an $(N-1)^{th}$-order Chebyshev approximation with coefficients β_0^i through β_{N-1}^i. The velocity state coefficients β_k are related to the integrand coefficients F_k^{i-1} by the expressions:

$$\beta_{N-1} = \frac{1}{2(N-1)} F_{N-2} \tag{D.78a}$$

$$\beta_{N-2} = \frac{1}{2(N-2)} F_{N-3} \tag{D.78b}$$

$$\beta_k = \frac{1}{2k}(F_{k-1} - F_{k+1}) \ , \quad k = 1, 2, .., N-3 \tag{D.78c}$$

$$\beta_0 = 2v_0 + 2\sum_{k=1}^{N-1} \left[(-1)^{k+1}\beta_k\right] \tag{D.78d}$$

The position states are approximated using an N^{th}-order Chebyshev approximation with coefficients α_0^i through α_N^i. The position state coefficients α_k are related to the velocity state coefficients β_k^i by the expressions:

$$\alpha_N = \frac{1}{N}\beta_{N-1} \tag{D.79a}$$

$$\alpha_{N-1} = \frac{1}{2(N-1)}\beta_{N-2} \tag{D.79b}$$

$$\alpha_k = \frac{1}{2k}(\beta_{k-1} - \beta_{k+1}) \ , \quad k = 1, 2, .., N-2 \tag{D.79c}$$

$$\alpha_0 = 2x_0 + 2\sum_{k=1}^{N-1} \left[(-1)^{k+1}\alpha_k\right] + \left[(-1)^{N+1}\alpha_N\right] \tag{D.79d}$$

Once again, in lieu of the power series basis functions, we could obtain Equations D.78 and D.79 by retaining the Chebyshev basis functions as in Equations D.47.

This more general development is accomplished by using Equations C.11 and C.12 to integrate $T_k(s)$ in the right side integrands of Equations D.47 and collecting the **R.H.S.** as series in $T_k(\tau)$. The **L.H.S.** of Equations D.47 is then equated to the **R.H.S.** coefficients of $T_k(\tau)$, and upon imposing the boundary conditions at $\tau = -1$, we obtain Equations D.78 and D.79 without introducing power series.

D.2.2 Vector/Matrix MCPI

Analogous to the first-order MCPI formulation of Section D.1, the equations derived in Section D.2.1 may be restructured into a vector/matrix framework. Although the method was derived for a scalar ODE system, we again generalize to a state vector of arbitrary length n. This means we have a position state vector $\boldsymbol{x}(t) \in \mathbb{R}^n$, and a velocity state vector $\dot{\boldsymbol{x}}(t) \in \mathbb{R}^n$.

As with the first-order method, the integrand of the ODE is fit using the least squares Chebyshev approximation of Equation D.26. A Chebyshev approximation of order $N-2$, with $M = N$ CGL sample points is used. During the i^{th} Picard iteration, the $(N-1) \times n$ matrix of integrand coefficients F^{i-1} contains the individual $n \times 1$ integrand coefficient vectors $\boldsymbol{F}_0^{i-1}$ through $\boldsymbol{F}_{N-2}^{i-1}$. The integrand coefficient fit may be written as a vector/matrix expression of the same form as Equations D.27, or

$$F^{i-1} = {}^{F}T^{T} V G(X^{i-1}, V^{i-1}) \tag{D.80}$$

for short. In the above equation, ${}^{F}T^{T}$ is an $(N-1) \times (M+1)$ matrix of Chebyshev polynomials T_0 through T_{N-2} evaluated at the $M+1$ CGL sample points, and V is an $(M+1) \times (M+1)$ Cheybshev weight matrix as defined in Equations D.27. $G(X^{i-1}, V^{i-1})$ is the result of evaluation of the integrand functions at the $(i-1)^{th}$ approximation of the system states, where we have defined the i^{th} estimate of the position and velocity states as a matrix by stacking the state estimate values for all

$(M+1)$ sampled times τ_0 to τ_N as

$$X^i = matrix\{\boldsymbol{x}^i(\tau_0)^T \ ; \ \boldsymbol{x}^i(\tau_1)^T \ ; \ ... \ ; \ \boldsymbol{x}^i(\tau_M)^T\} \tag{D.81a}$$

$$V^i = matrix\{\dot{\boldsymbol{x}}^i(\tau_0)^T \ ; \ \dot{\boldsymbol{x}}^i(\tau_1)^T \ ; \ ... \ ; \ \dot{\boldsymbol{x}}^i(\tau_M)^T\} \tag{D.81b}$$

where each $\boldsymbol{x}^i(\tau_j) = [x_1^i(\tau_j) \ , \ x_2^i(\tau_j) \ , \ ... \ , \ x_n^i(\tau_j)]^T$ and $\dot{\boldsymbol{x}}^i(\tau_j) = [\dot{x}_1^i(\tau_j) \ , \ \dot{x}_2^i(\tau_j) \ , \ ... \ , \ \dot{x}_n^i(\tau_j)]^T$ is an $(n \times 1)$ vector. The slightly confusing notation for the weight matrix V and the i^{th} velocity state approximation V^i is being used in order remain consistent with previous MCPI publications.

Summarizing Equations D.78, the Picard iteration to relate the unknown velocity state coefficients β^i to the known integrand coefficients F^{i-1} may be written as Equations D.82 and D.83. Similarly, Equations D.79 relating the unknown position state coefficients α^i to the (now) known velocity state coefficients β^i may be written as Equations D.84 and D.85.

$$\beta^i = \begin{bmatrix} (\boldsymbol{\beta}_0^i)^T \\ (\boldsymbol{\beta}_1^i)^T \\ \vdots \\ (\boldsymbol{\beta}_k^i)^T \\ \vdots \\ (\boldsymbol{\beta}_{N-2}^i)^T \\ (\boldsymbol{\beta}_{N-1}^i)^T \end{bmatrix} = \begin{bmatrix} 2\boldsymbol{v_0}^T + 2\sum_{k=1}^{N-1}\left[(-1)^{k+1}\boldsymbol{\beta}_k^i\right]^T \\ \frac{1}{2}(\boldsymbol{F}_0^{i-1} - \boldsymbol{F}_2^{i-1})^T \\ \vdots \\ \frac{1}{2k}(\boldsymbol{F}_{k-1}^{i-1} - \boldsymbol{F}_{k+1}^{i-1})^T \\ \vdots \\ \frac{1}{2(N-2)}(\boldsymbol{F}_{N-3}^{i-1})^T \\ \frac{1}{2(N-1)}(\boldsymbol{F}_{N-2}^{i-1})^T \end{bmatrix} \tag{D.82}$$

$$\beta^i = \begin{bmatrix} 2\boldsymbol{v_0}^T \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \vdots \\ \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix} + \begin{bmatrix} 1 & & & & & & \\ & \frac{1}{2} & & & & & \\ & & \ddots & & & & \\ & & & \frac{1}{2k} & & & \\ & & & & \ddots & & \\ & & & & & \frac{1}{2(N-2)} & \\ & & & & & & \frac{1}{2(N-1)} \end{bmatrix} \begin{bmatrix} 1 & -\frac{1}{2} & S(1,3) & \dots & S(1,j) & \dots & S(1,N-1) \\ 1 & 0 & -1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & -1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} (\boldsymbol{F}_0^{i-1})^T \\ (\boldsymbol{F}_1^{i-1})^T \\ (\boldsymbol{F}_2^{i-1})^T \\ \vdots \\ (\boldsymbol{F}_{N-4}^{i-1})^T \\ (\boldsymbol{F}_{N-3}^{i-1})^T \\ (\boldsymbol{F}_{N-2}^{i-1})^T \end{bmatrix} \tag{D.83}$$

$$\alpha^i = \begin{bmatrix} (\boldsymbol{\alpha}_0^i)^T \\ (\boldsymbol{\alpha}_1^i)^T \\ \vdots \\ (\boldsymbol{\alpha}_k^i)^T \\ \vdots \\ (\boldsymbol{\alpha}_{N-1}^i)^T \\ (\boldsymbol{\alpha}_N^i)^T \end{bmatrix} = \begin{bmatrix} 2\boldsymbol{x}_\mathbf{0}^T + 2\sum_{k=1}^{N-1}\left[(-1)^{k+1}\boldsymbol{\alpha}_k^i\right]^T + \left[(-1)^{N+1}\boldsymbol{\alpha}_N^i\right]^T \\ \frac{1}{2}(\boldsymbol{\beta}_0^i - \boldsymbol{\beta}_2^i)^T \\ \vdots \\ \frac{1}{2k}(\boldsymbol{\beta}_{k-1}^i - \boldsymbol{\beta}_{k+1}^i)^T \\ \vdots \\ \frac{1}{2(N-1)}(\boldsymbol{\beta}_{N-2}^i)^T \\ \frac{1}{N}(\boldsymbol{\beta}_{N-1}^i)^T \end{bmatrix} \tag{D.84}$$

$$\alpha^i = \begin{bmatrix} 2\boldsymbol{x}_\mathbf{0}^T \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} + \begin{bmatrix} 1 & & & & & & \\ & \frac{1}{2} & & & & & \\ & & \ddots & & & & \\ & & & \frac{1}{2k} & & & \\ & & & & \ddots & & \\ & & & & & \frac{1}{2(N-1)} & \\ & & & & & & \frac{1}{N} \end{bmatrix} \begin{bmatrix} 1 & -\frac{1}{2} & S(1,3) & \dots & S(1,j) & \dots & S(1,N) \\ 1 & 0 & -1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & -1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix} \begin{bmatrix} (\boldsymbol{\beta}_0^i)^T \\ (\boldsymbol{\beta}_1^i)^T \\ (\boldsymbol{\beta}_2^i)^T \\ \vdots \\ (\boldsymbol{\beta}_{N-3}^i)^T \\ (\boldsymbol{\beta}_{N-2}^i)^T \\ (\boldsymbol{\beta}_{N-1}^i)^T \end{bmatrix} \tag{D.85}$$

Equations D.83 and D.85 may be written more compactly as

$$\beta^i = V_0 + {}^{\beta}R^{\beta}SF^{i-1} \tag{D.86a}$$

$$\alpha^i = X_0 + {}^{\alpha}R^{\alpha}S\beta^i \tag{D.86b}$$

V_0 and X_0 are of size $N \times n$ and $(N+1) \times n$, respectively, and are responsible for enforcing the initial boundary conditions for the velocity and position coefficients. ${}^{\beta}R$ and ${}^{\alpha}R$ are square matrices of size $N \times N$ and $(N+1) \times (N+1)$, respectively. These two weight matrices reflect the difference between using the least squares Chebyshev formulation for both sets of states (as with β^i and F^{i-1}), and using the least squares Chebyshev formulation for one set and the interpolation formulation for the other other (as in the the case of α^i and β^i), as discussed in Section D.2.1. ${}^{\beta}S$ and ${}^{\alpha}S$ are of size $N \times (N-1)$ and $(N+1) \times N$, respectively. Looking at the explicit expansions in Equations D.62b and D.77b of the previous section, it is easily verified that the general form of the top row of the ${}^{\beta}S$ and ${}^{\alpha}S$ matrices are

$$S(1,k) \equiv (-1)^k \left(\frac{1}{k-2} - \frac{1}{k} \right) \tag{D.87}$$

which is the same result as for the first-order MCPI case.

Having found expressions for the velocity and position coefficients, we may now define the velocity and position states themselves. The velocity states are expressed

as

$$V^i = \begin{bmatrix} T_0(\tau_0) & T_1(\tau_0) & \dots & T_{N-1}(\tau_0) \\ T_0(\tau_1) & T_1(\tau_1) & \dots & T_{N-1}(\tau_1) \\ \vdots & \vdots & \dots & \vdots \\ T_0(\tau_M) & T_1(\tau_M) & \dots & T_{N-1}(\tau_M) \end{bmatrix} \begin{bmatrix} \frac{1}{2} & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \\ & & & & \frac{1}{2} \end{bmatrix} \begin{bmatrix} (\boldsymbol{\beta_0^i})^T \\ \vdots \\ (\boldsymbol{\beta_k^i})^T \\ \vdots \\ (\boldsymbol{\beta_{N-1}^i})^T \end{bmatrix} \tag{D.88}$$

or, in shorthand

$$V^i = {}^{\beta}T\,{}^{\beta}W\,\beta^i \tag{D.89}$$

${}^{\beta}W$ is an $N \times N$ diagonal weight matrix, and ${}^{\beta}T$ is a matrix of Chebyshev polynomials evaluated at CGL sampled times τ_j, in this case ${}^{\beta}T$ is of size $(M+1) \times N$.

Similarly, the position states are

$$X^i = \begin{bmatrix} T_0(\tau_0) & T_1(\tau_0) & \dots & T_N(\tau_0) \\ T_0(\tau_1) & T_1(\tau_1) & \dots & T_N(\tau_1) \\ \vdots & \vdots & \dots & \vdots \\ T_0(\tau_M) & T_1(\tau_M) & \dots & T_N(\tau_M) \end{bmatrix} \begin{bmatrix} \frac{1}{2} & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \\ & & & & \frac{1}{2} \end{bmatrix} \begin{bmatrix} (\boldsymbol{\alpha_0^i})^T \\ \vdots \\ (\boldsymbol{\alpha_k^i})^T \\ \vdots \\ (\boldsymbol{\alpha_N^i})^T \end{bmatrix} \tag{D.90}$$

or, in shorthand

$$X^i = {}^{\alpha}T\,{}^{\alpha}W\,\alpha^i \tag{D.91}$$

${}^{\alpha}W$ is an $(N+1)\times(N+1)$ diagonal weight matrix, and ${}^{\alpha}T$ is of size $(M+1)\times(N+1)$. Note that ${}^{\alpha}T$ has a $\frac{1}{2}$ as the last term, whereas in ${}^{\beta}T$ the last term is a 1. This is caused by using the Chebyshev interpolation formulation for the position states, but using the Chebyshev least squares formulation for the velocity states.

Summarizing the expressions so far, we have

$$F^{i-1} = {}^{F}T^{T}VG(X^{i-1}, V^{i-1}) \tag{D.92a}$$

$$\beta^{i} = V_0 + {}^{\beta}R^{\beta}SF^{i-1} \tag{D.92b}$$

$$\alpha^{i} = X_0 + {}^{\alpha}R^{\alpha}S\beta^{i} \tag{D.92c}$$

$$V^{i} = {}^{\beta}T^{\beta}W\beta^{i} \tag{D.92d}$$

$$X^{i} = {}^{\alpha}T^{\alpha}W\alpha^{i} \tag{D.92e}$$

Defining constant matrices

$$C_\alpha \equiv RS^{F}T^{T}V \tag{D.93a}$$

$$C_x \equiv TW \tag{D.93b}$$

$$C_\gamma \equiv RS \tag{D.93c}$$

a Picard update is of the form

$$F^{i-1} = {}^{F}T^{T}VG(X^{i-1}, V^{i-1}) \tag{D.94a}$$

$$\beta^{i} = V_0 + {}^{\beta}C_{\alpha}G^{i-1} \tag{D.94b}$$

$$\alpha^{i} = X_0 + {}^{\alpha}C_{\gamma}\beta^{i} \tag{D.94c}$$

$$V^{i} = {}^{v}C_{x}\beta^{i} \tag{D.94d}$$

$$X^{i} = {}^{x}C_{x}\alpha^{i} \tag{D.94e}$$